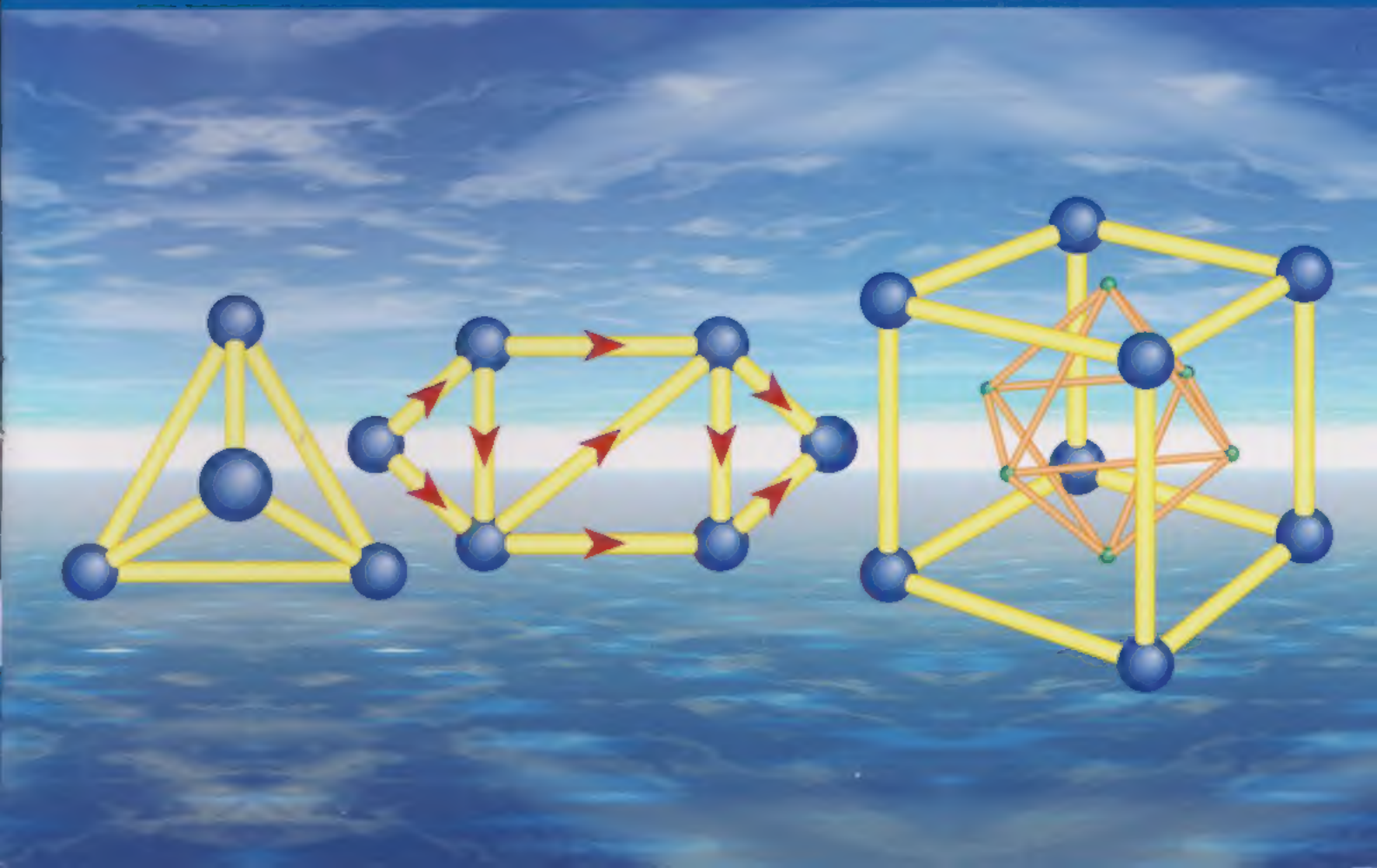
The Open University

MT365 Graphs, Networks and Design

Introduction



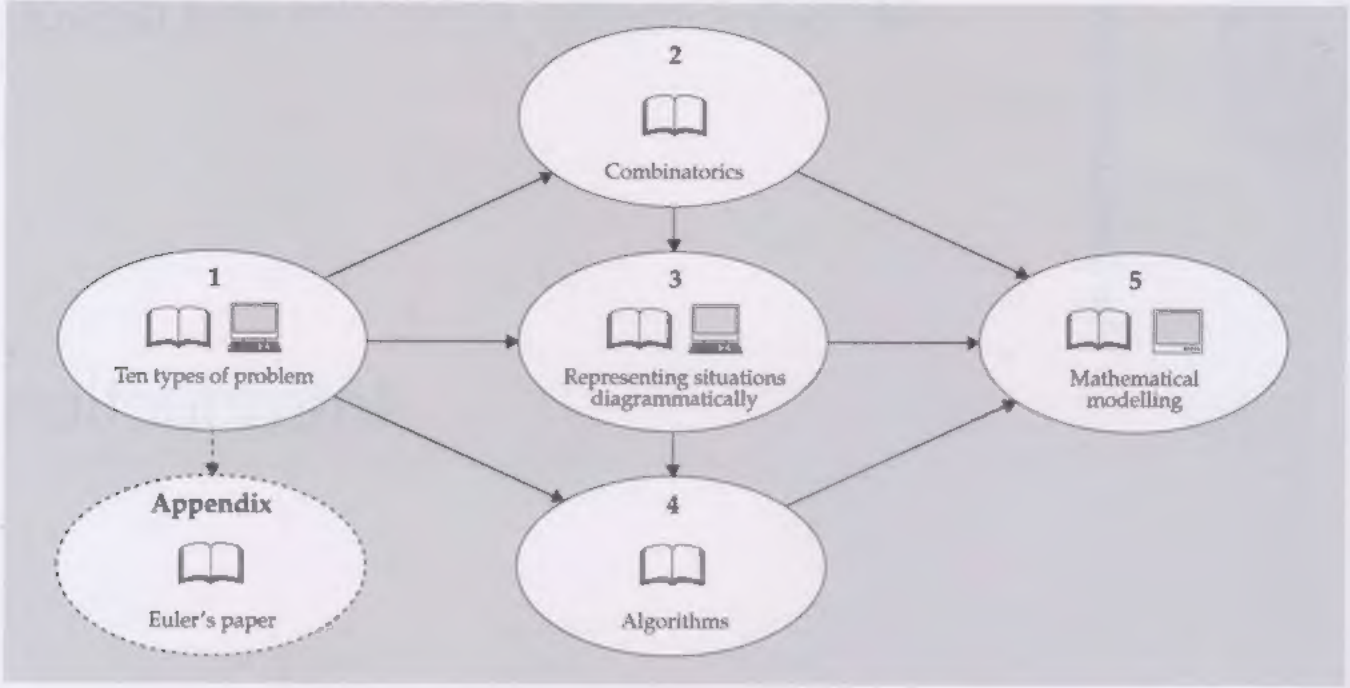
The Open University

MT365 Graphs, networks and design

Introduction

Study guide

Before you begin this unit, ensure that you have read the *Course Guide* and have worked through the *Introduction to the Course Software*.



In Sections 1 and 3, you will start to develop techniques for solving a wide range of problems. You should try each of the problems we set you, but do not spend too long on any one. These two sections also ask you to use your computer to explore some of the problems, by means of activities described in the *Computer Activities Booklet*. The activities are best undertaken either immediately after you have studied the corresponding problem, or at the end of that section, as indicated in the text. **Do not attempt any of the activities until you have worked through the *Introduction to the Course Software*.**

In Sections 2, 4 and 5 we introduce the main themes of the course. In these three sections, don't get bogged down in the details — your aim should simply be to appreciate the main ideas.

The Appendix is optional, and is included for interest only. It may be read any time after you have studied Section 1.4.

There is a television programme associated with Section 5.

There is no audio-tape associated with this unit.

The Open University, Walton Hall, Milton Keynes, MK7 6AA.

First published 1995. Reprinted 1998, 2002, 2003, 2005, 2008, 2009.

Printed and bound by Page Bros, Norwich.

ISBN 0 7492 2224 7

1.5

Contents

Introduction ALL NOT DIRECTLY ASSESSED.

In this introductory unit, we aim to give you some idea of what this course is about. You will meet a number of types of problem, instances of which you will be invited to try to solve. Some of these are straightforward, and are included to give you an idea of some of the topics in this course. Others are more difficult and illustrate the need for a more systematic approach, to be given later in the course.

We start, in Section 1, *Ten types of problem*, by describing ten types of problem that will appear later in the course. Some of these are theoretical in nature, whereas others are simple instances of practical problems. Later in the course, we shall return to them when we have more techniques at our disposal.

In Section 2, *Combinatorics*, we introduce one of the main themes of the course. After explaining what is meant by the term *combinatorics*, we show how combinatorial problems can be classified into four classes — existence, construction, enumeration and optimization problems — and give several examples from each class. This classification is often useful in helping us to identify the type of solution we are seeking for any particular problem.

In Section 3, *Representing situations diagrammatically*, we return to some of the problems introduced in Section 1, and show that an important step in the process of solving a problem may be a diagrammatic representation of the situation.

In order to construct the solution to a given type of problem, we may need to find or develop a step-by-step procedure. Such a procedure is called an *algorithm*, and is discussed in Section 4, *Algorithms*. Here we examine some specific algorithms, and this leads to a brief discussion of the efficiency of algorithms.

In Section 5, *Mathematical modelling*, we describe the *modelling process*, in which the main features of a practical problem are expressed in mathematical terms as faithfully as possible, the resulting mathematical problem is solved, and the mathematical solution is then interpreted in its original setting. The television programme for this unit forms part of this section; it considers the modelling of a particular problem, that of determining the optimum locations of fire stations.

1 Ten types of problem

In this section, we describe ten types of problem that you will meet in this course. Most of them are necessarily expressed in rather simple form, since we have as yet no appropriate technical language that we can use to describe them and no suitable techniques with which to solve them. All of them are instances of more substantial topics that you will study later.

You should try all the problems we set you in order to get a feeling for what they involve, but do not spend too much time on any one of them if you get bogged down. Several of them are designed to show you that a simple-minded approach is not always sufficient, and that more systematic methods may be needed. You will meet such methods later in the course.

Associated with each of the first six types of problem are short computer activities; you can either work through each computer activity immediately after studying the corresponding text, or you can work

through all of them at the end of the section. In any case, do not spend long on any activity.

Finally, note that several of the problems in this section are designed to be open-ended and exploratory, and that you will not find solutions to all of them in this unit.

> And suppose we solve all the problems it presents? What happens? We end up with more problems than we started with. Because that's the way problems propagate their species. A problem left to itself dries up or goes rotten. But fertilize a problem with a solution — you'll hatch out dozens.
>
> N. F. Simpson, *A Resounding Tinkle.*

1.1 Map colouring

Consider the following map of the USA (excluding Alaska and Hawaii):

It is common for maps of this kind to be coloured in such a way that neighbouring states (or countries) are coloured differently. This enables us to distinguish easily between the various states, and to locate the state boundaries. The question arises:

> how many colours are needed to colour the entire map?

Note that the larger and more complicated a map, the more colours we might expect to need to colour it.

Problem 1.1

Can the above map of the USA be coloured with just three colours?

Hint Consider Nevada and its neighbouring states.

The solution to Problem 1.1 shows that there are some maps that cannot be coloured with just three colours. So, not *every* map can be coloured with just three colours. But can every map be coloured with just four colours? The computer activity related to this subsection invites you to explore this question.

Map colouring problems are discussed further in *Graphs 3, Planarity and colouring.*

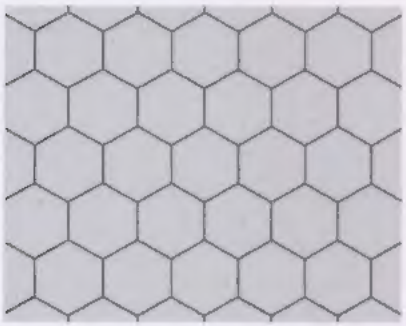
If you wish, you can now proceed directly to the computer activity on **Map colouring**. (All the computer activities are described in the *Computer Activities Booklet.*)

1.2 Tilings

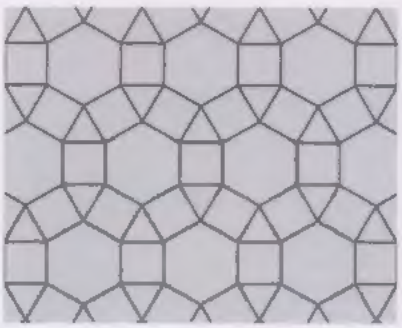
If we attempt to tile a flat surface with tiles, we find that only certain shapes and arrangements are possible. Given a supply of tiles of assorted sizes and shapes, we cannot guarantee that they will all fit together neatly without gaps or overlaps. However, if all the tiles are regular polygons of the same size and shape, then we can determine whether such a tiling is possible.

In a *regular polygon*, all sides have equal length and all interior angles are equal; examples of regular polygons are equilateral triangles and squares.

Tiling (a) below is a tiling with regular hexagons. Note that the tiles fit together without gaps or overlaps, and that the sides of neighbouring polygons match up exactly. Also, the arrangement of hexagons around each corner is the same. This tiling can be extended as far as we wish in all directions. Such a tiling with regular polygons is called a **regular tiling**.



(a)



(b)

We can also construct tilings from regular polygons of two or more different types. For example, tiling (b) above is constructed from equilateral triangles, squares and regular hexagons. Again, the arrangement of the polygons around each corner is the same. Such a tiling is called a **semi-regular tiling**.

Problem 1.2

(a) Construct a regular tiling consisting entirely of equilateral triangles.

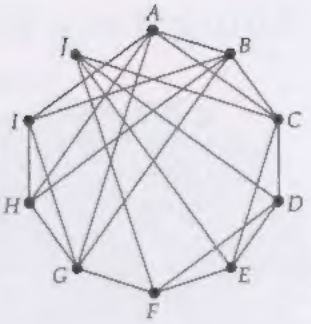
(b) Construct a semi-regular tiling consisting entirely of squares and regular octagons (eight-sided polygons).

You have seen above and in Problem 1.2 that regular tilings of equilateral triangles and of regular hexagons exist, as do semi-regular tilings of squares and regular octagons and of equilateral triangles, squares and regular hexagons. But which other regular polygons or combinations of regular polygons can be used to construct regular or semi-regular tilings? The related computer activities invite you to explore this question.

Tilings are discussed further in *Design 1, Geometric design.*

If you wish, you can now proceed directly to the computer activities on **Tilings**.

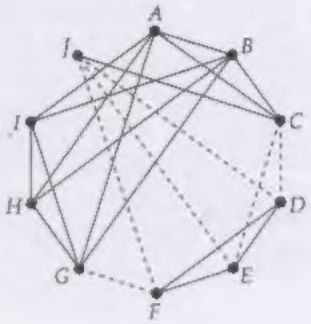
1.3 Connection problems



The diagram in the margin represents a telecommunication network. The points labelled *A–J* represent telephone exchanges, and the lines represent links connecting them in pairs — for example, a link may be a cable, an optical fibre or a radio link via a satellite.

Such a telecommunication network is vulnerable in two ways: some of the links may become damaged and put out of action, or some of the exchanges (and their associated links) may be put out of action (for example, by accidents or earthquakes).

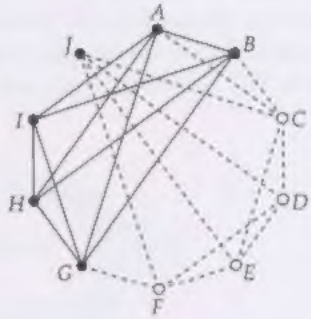
Suppose that, at times, some of the *links* in the above network are out of action; this may make connections (direct or indirect) between some exchanges impossible. For example, if the links *FG*, *FJ*, *EJ*, *DJ*, *CE* and *CD* are all out of action, then the part of the diagram containing the exchanges *D*, *E* and *F* is cut off from the rest. We are interested in the following questions:



what is the *smallest* number of links whose closure would effectively separate the network into two parts, so that no exchange in one part can communicate with any exchange in the other part?

which are the corresponding links?

At other times, some of the *exchanges* may be out of action, and so connections between the remaining exchanges cannot be made via the links connected to these exchanges; this may make connections between some exchanges impossible. For example, if the exchanges *C*, *D*, *E* and *F* are all out of action, then the part of the diagram containing the exchange *J* is cut off from the rest. We are interested in the following questions:



what is the *smallest* number of exchanges whose closure would effectively separate the network into two (or more) parts?

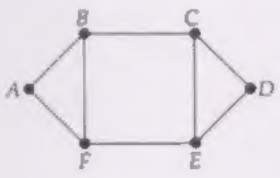
which are the corresponding exchanges?

Before using your computer to try to answer these questions for the above network, try the following problem.

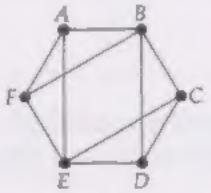
Connection problems are discussed further in *Networks 1, Network flows.*

Problem 1.3

The following diagrams represent telecommunication networks:



(a)



(b)

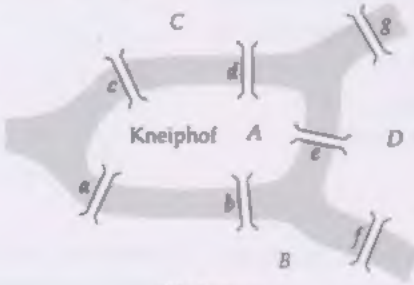
In each case, find:

- the smallest number of *links* whose closure would separate the network into two parts, and the corresponding links;
- the smallest number of *exchanges* whose closure would separate the network into two parts, and the corresponding exchanges.

If you wish, you can now proceed directly to the computer activities on **Connection problems.**

1.4 Königsberg bridges

In the early eighteenth century the mediaeval city of Königsberg in Eastern Prussia contained a central island called Kneiphof, around which the river Pregel flowed before dividing into two. The four parts of the city (*A*, *B*, *C* and *D*) were interconnected by seven bridges (*a*, *b*, *c*, *d*, *e*, *f* and *g*), as shown in the following diagram:



Königsberg

It is said that the citizens of Königsberg entertained themselves by trying to find a route that crosses each bridge exactly once and ends at the starting point. Try as they might, they could find no such route, and they began to believe the task impossible.

Problem 1.4

In the above Königsberg bridges diagram, try to find a route that crosses each bridge exactly once and ends at the starting point. Do you think that such a route exists?

Historical note

Leonhard Euler (1707–1783), possibly the most prolific mathematician of all time, solved the Königsberg bridges problem in an important paper entitled *Solutio problematis ad geometriam situs pertinentis* (The solution of a problem relating to the geometry of position). The part of this paper that relates directly to the Königsberg bridges problem is given in the Appendix.

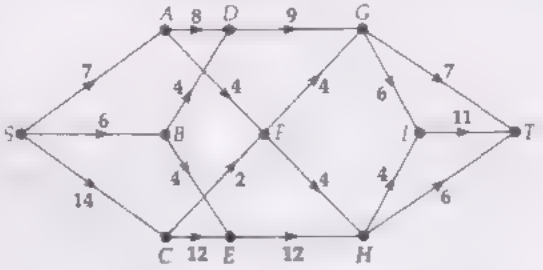
The related computer activities ask you to try to suggest a rule for solving the Königsberg bridges problem and similar problems.

The Königsberg bridges problem and similar problems are discussed further in Section 3 and in *Graphs 1, Graphs and digraphs*.

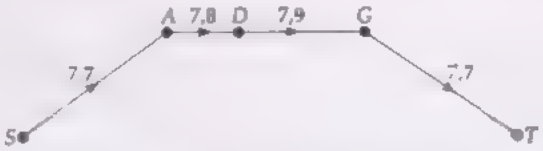
If you wish, you can now proceed directly to the computer activities on **Königsberg bridges**.

1.5 Network flows

The following diagram represents a network of pipelines along which a fluid (for example, gas, oil or water) flows from a starting point S to a terminal T. Each of the intermediate points A–I represents a pipe junction at which the total flow into the junction must equal the total flow out (so that no fluid is 'lost' along the way). Each line between two junctions represents a pipeline, and the number next to it is the *capacity* of that pipeline (in some appropriate units); the flow along a pipeline must not exceed its capacity, and must be in the direction indicated.



Inspection of the above diagram shows that a *flow* of at most 7 units of fluid can be sent along the route $SADGT$ without exceeding the capacity of any of the pipelines SA, AD, DG or GT. This is illustrated in the following diagram, where the first number on each line represents the flow along that pipeline and the second number — in bold — its capacity.



In order to send more fluid from S to T, other routes must be used as well.

Problem 1.5

(a) How can 13 units of fluid be sent from S to T without exceeding the capacity of any pipeline?

(b) How can 15 units of fluid be sent?

You have seen how 7, 13 and 15 units of fluid can be sent from S to T. But what is the *maximum flow* from S to T — that is, what is the maximum amount of fluid that can be sent from S to T at any one time? The related computer activity asks you to try to find the maximum flow.

Network flows are discussed further in *Networks 1, Network flows.*

If you wish, you can now proceed directly to the computer activity on **Network flows.**

1.6 Braced rectangular frameworks

Many buildings are supported by rectangular steel frameworks, and it is important that such frameworks should remain rigid under heavy loads. One way to achieve this is to add *braces*, to prevent distortion.

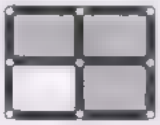
For example, the following diagram shows how a simple unbraced rectangular framework can be distorted.

non-rigid

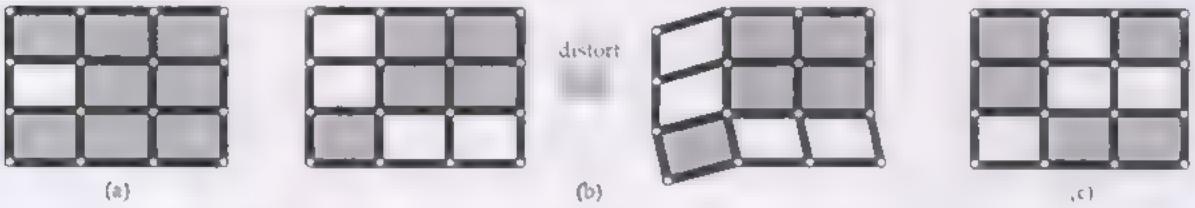
Now, adding only two braces, in the form of rectangular plates (indicated by shading), cannot make this framework rigid, as the following diagrams illustrate.

non-rigid non-rigid

The minimum number of braces that we must add to make this framework rigid is three.



Now consider the following three frameworks:



(a) (b) (c)

(a) This framework is rigid, but is over-braced, since some braces can be removed while maintaining its rigidity.

(b) This framework is not rigid, since it can be distorted as shown.

(c) Is this framework rigid? If so, can any braces be removed while maintaining its rigidity? If not, how can it be made rigid?

Problem 1.6

(a) Which braces can be removed from framework (a) while maintaining its rigidity?

(b) Which braces can be added to framework (b) to make it rigid?

(c) Try to answer the questions above relating to framework (c).

You may have found Problem 1.6 quite difficult, because the possible distortions are hard to visualize. The related computer activity enables you to *see* how non-rigid frameworks can be distorted, and so helps you to explore the above ideas further by asking you to choose possible bracings and observe the possible distortions.

Braced rectangular frameworks are discussed further in Section 3 and in *Design 2, Kinematic design*.

If you wish, you can now proceed directly to the computer activity on **Braced rectangular frameworks**.

1.7 Job assignment

A building contractor advertises five jobs — those of bricklayer, carpenter, decorator, electrician and plumber. There are four applicants — one for carpenter and decorator, one for bricklayer, carpenter and plumber, one for decorator, electrician and plumber, and one for carpenter and electrician. Can all of the applicants be assigned to jobs for which they are qualified?

In order to solve this problem, it is convenient to represent the information in tabular form, as shown below:

applicant	job
1	*c, d*
2	*b, c, p*
3	*d, e, p*
4	*c, e*

From the table, we can see that one possible assignment of applicants to jobs is:

1–carpenter,

2–plumber,

3–decorator,

4–electrician.

Problem 1.7

(a) The above is not the only solution; list as many other solutions as you can.

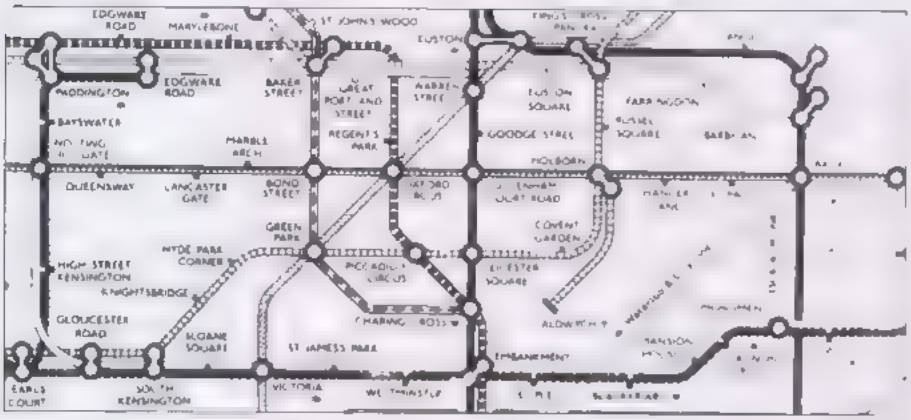
(b) Suppose that applicant 2 decides not to apply for the position of plumber. Is it still possible to assign the four applicants to jobs for which they are qualified? If so, how can this be done? If not, how many of the positions can be filled?

Such problems can become extremely complicated when there are large numbers of applicants and jobs. Moreover, some applicants may be more suited to particular jobs than others, and we may need to take this into consideration.

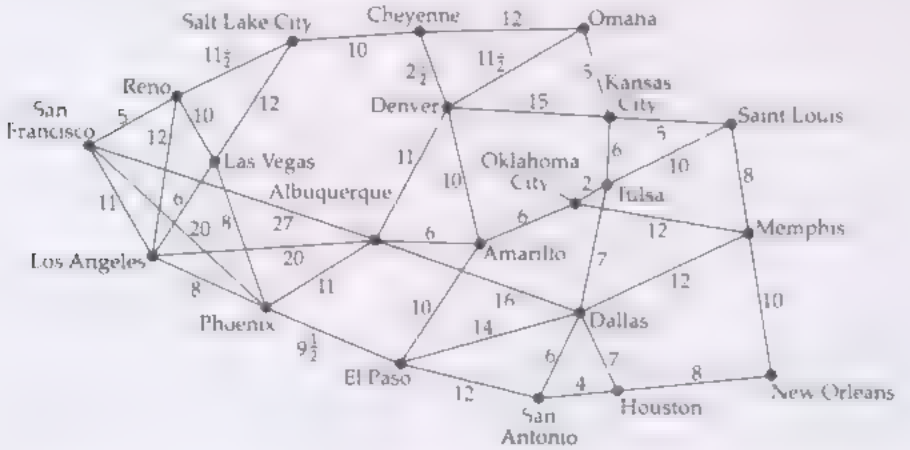
Assignment problems are discussed further in *Networks 3, Assignment and transportation.*

1.8 Optimal routes

The following diagram represents the central part of the London Underground. Like all maps, it represents only those features that are relevant to its purpose. In the case of the London Underground map, it is the interconnections between stations that are important, not their precise geographical locations — and so the map emphasizes interconnections at the expense of precise geographical information.



However, when we use a road map, not only are the interconnections between towns important, but so are the distances or travel times between them. For example, the following road map shows some of the major routes between a number of cities in the USA, where the numbers indicate the travel times (in hours) between pairs of cities.



Problem 1.8

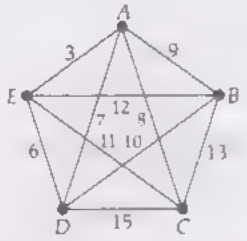
(a) What interpretations can be given to the word 'best' applied to routes between stations on the London Underground? For each of your interpretations, use the map above to find the 'best' route between Marble Arch and Westminster.

(b) In the case of the USA road map above, the most obvious interpretation of 'best' route is 'the route that takes the shortest travel time'. With this interpretation, use the map to find the 'best' route from Los Angeles to Amarillo, and from San Francisco to Denver.

The determination of 'best' routes in a variety of situations is discussed further in *Networks 2, Optimal paths.*

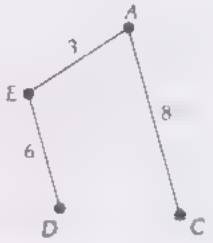
1.9 Minimum connector problems

Consider the case of an electricity company that wants to lay a network of cables in order to link together five towns, *A*, *B*, *C*, *D* and *E*. It wants to minimize the amount of cabling, in order to keep its costs down. The distances (in miles) between the towns are shown in the following diagram:

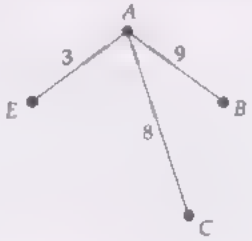


The electricity company's problem is one of finding a *minimum connector* — a set of links of minimum *total* length that connects all five towns.

For example, a minimum connector that links the towns *A*, *C*, *D* and *E* (but not *B*) comprises the links *AC*, *AE* and *DE*. This minimum connector has total length 17 miles.



Similarly, a minimum connector that links the towns *A*, *B*, *C* and *E* (but not *D*) comprises the links *AB*, *AC* and *AE*, of total length 20 miles.



Problem 1.9

(a) Find a minimum connector that links the towns *B*, *C*, *D* and *E*.

(b) Try to find a minimum connector that links all five towns.

Minimum connector problems are discussed further in Section 4 and in *Graphs 2, Trees*

1.10 Travelling salesman problems

A travelling salesman wishes to visit a number of cities and return to his starting point, selling his wares as he goes. He wants to select the route with the least total length. Which route should he choose? And what is its length?

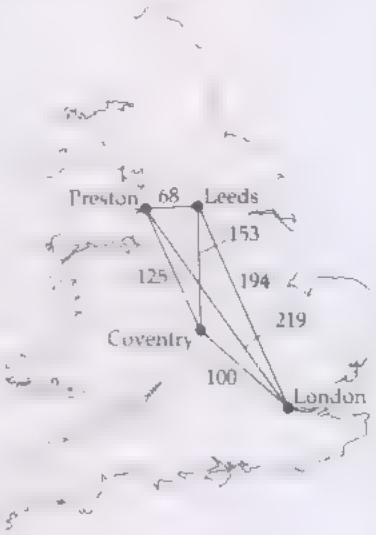
In principle, we can solve such problems by looking at all possible routes and choosing one with the least total length. For example, for the four cities on the map in the margin, trial-and-error methods show that a solution of the travelling salesman problem is the route

London – Coventry – Preston – Leeds – London

(in either direction), with total length 100 + 125 + 68 + 194 = 487 miles. Any other route would be longer. But this trial-and-error approach becomes difficult if the number of cities increases significantly.

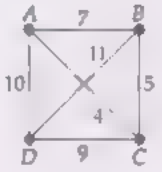
Unlike minimum connector problems, for which there are *efficient* systematic procedures for finding solutions — that is, systematic procedures that provide a solution quickly, in practice — there is no known efficient systematic procedure for travelling salesmen problems. Indeed, the only known procedure that is *guaranteed* to solve *any* given travelling salesman problem is the *exhaustion* method, which comprises looking at *all* possible routes and choosing the shortest. This is feasible if there are ten cities, since the number of possible routes is then $9! = 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 362\,880$, and a computer sorting through these at the rate of one thousand per second would find the shortest route in about six minutes. On the other hand, if there are twenty cities, then the number of possible routes is about 1.22×10^{20}, and a computer sorting through them at the same rate would take almost four million years! This rapid increase in the number of possible routes and in the time taken to find the shortest route, as the number of cities grows, are examples of a phenomenon known as the *combinatorial explosion*.

The travelling salesman problem first appeared in rudimentary form in a practical German book of 1831 for the *Handlungsreisende* (travelling salesman). Its first appearance in mathematical circles was at Princeton University in the 1930s.

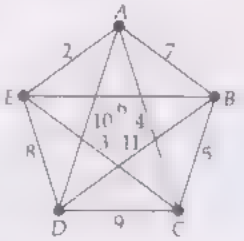


Problem 1.10

(a) A zoo-keeper, currently tending the antelopes, also needs to tend the bears, camels and dingos, before returning to the antelopes. The distances (in tens of metres) between the enclosures for the four types of animal are as shown in the margin. Find a route with the least total length.



(b) The zoo has suffered staff cut-backs, and so the zoo-keeper now needs to include the elephants on his route. The distances between the five enclosures are as shown in the margin. Try to find a new route with the least total length.



Although there is no known efficient procedure that is guaranteed to solve *any* given travelling salesman problem, several efficient procedures are known that give an *approximation* to the minimum length of a route that visits all the cities and returns to the starting point. Such a procedure is presented in Section 4.

Travelling salesman problems are discussed further in *Graphs 2, Trees.*, and *Graphs 4, Graphs and computing*

If you have not already done so, you should now carry out the computer activities for Section 1.

* Since each route can be traced in either direction, (Ernie)
(In addition, halve the numbers to get $\frac{1}{2} \times 9! = 181440$ is about 3 minutes, and 6.08×10^{10} is roughly 2 million years.)

After studying this section, you should be able to explain what are meant by:

- map colourings;
- tilings;
- connection problems;
- the Konigsberg bridges problem (and similar problems);
- network flows;
- braced rectangular frameworks;
- job assignment problems;
- optimal routes,
- minimum connectors;
- travelling salesman problems.

2 Combinatorics

In this section we discuss one of the main themes of the course —*combinatorics*. We attempt to explain what combinatorics is, and what types of problem it is concerned with, and we show how certain problems arising in technology and science can be represented and solved using combinatorial ideas and techniques.

2.1 What is combinatorics?

Although this is such a simple question to ask, it is very difficult to give a precise answer, since the term *combinatorics* means different things to different people. According to the *Encyclopaedia Britannica*:

> The scope of combinatorics is hard to define with any exactitude. In general, however, it may be said that it is concerned with arrangements, operations and selections within a finite or a discrete system.

In contrast, the *Oxford English Dictionary* describes the word 'combinatorial' as

> of or relating to (mathematical) combinations,

whereas *Webster's Dictionary* defines 'combinatorial analysis' as

> the general doctrine of the distribution of objects into classes.

None of these descriptions is really satisfactory; however, they indicate that combinatorics is concerned with *the arrangement, classification and enumeration of objects of various kinds.*

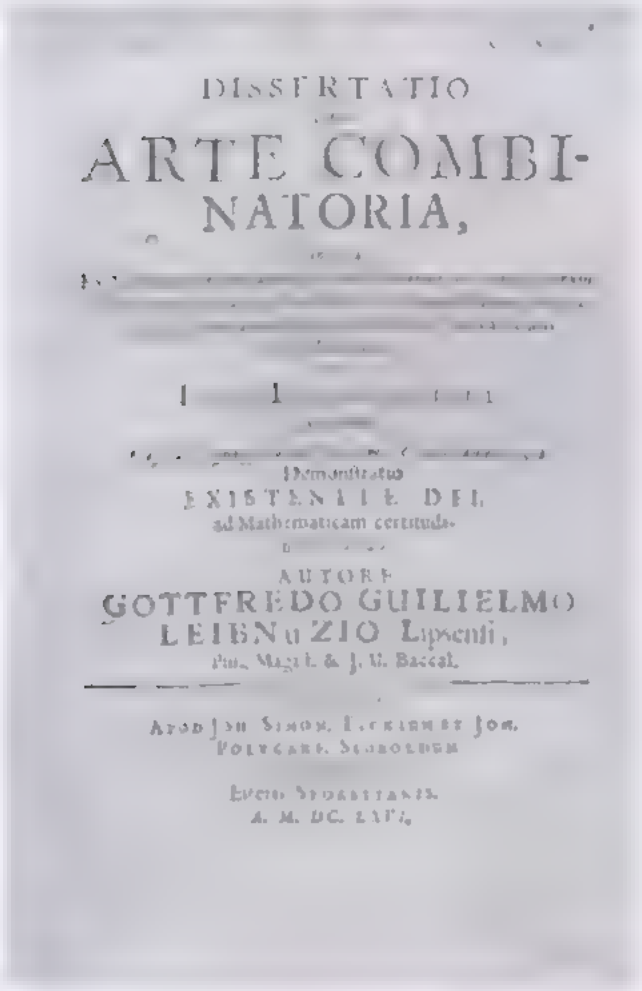
If one phrase deserves to be singled out from the above descriptions as characteristic of the subject, it is the phrase *within a finite or a discrete system*. Throughout this course, our concern is primarily with finite sets of objects, and the word *set* or *collection* is usually taken to mean *finite set*, unless otherwise stated. Similarly, we shall primarily be concerned with numerical quantities that change *discretely* (that is, in separate steps) rather than in a continuous way.

Historical note

The origins of combinatorics can be traced back to problems involving permutations and combinations in ancient India and China. The term 'combinatorial' first appeared in the Middle Ages, and appears in a number of theological works throughout the Renaissance. The following illustrations show two title pages from the 1660s.

The first is from a short treatise *Dissertatio de Arte Combinatoria* by the German mathematician and philosopher Gottfried Wilhelm Leibniz (1646–1716), who described the 'art of combinatorics' as the study of placing, ordering and choosing a number of objects. In his treatise, Leibniz claims to 'prove the existence of God with complete mathematical certainty'; we make no such claims for this course!

The second is from the book *Ars Magna Sciendi sive Combinatoria* by the Jesuit priest Athanasius Kircher (1601–1680), in which he attempts to unify all knowledge into a single system based on combinatorial principles.



DISSERTATIO

ARTE COMBI-
NATORIA,

Demonstratio

EXISTENTIÆ DEI

ad Mathematicam certitudinem

AUTORE

GOTTFREDO GUILIELMO

LEIBNUTIO Lipsiensi,

Phil. Magist. & J. U. Baccal.

In this course we take a rather different view of the subject from that of some of the sources cited above. Although many problems in combinatorics can be classified as problems in graphs, networks and/or design, it is also useful to consider a different classification based on four types of question that commonly arise. Most of the combinatorial problems you will meet can be described under one or more of the following interrelated headings:

Existence problems does there exist ... ? is it possible to ... ?

Construction problems if ... exists, how can we construct it?

Enumeration problems how many ... are there? can we list them all?

Optimization problems if there are several ... , which is the best?

Before discussing each of these classes of problem in detail, we interpret them in the context of two examples that you met in Section 1.

Example 2.1: braced rectangular frameworks

In investigating a given rectangular framework, we might wish to consider any of the following questions:

Existence problem	is it possible to brace the framework so as to make it rigid?
Construction problem	if such a bracing exists, how can we construct one?
Enumeration problem	how many rigid bracings are there, and can we list them all?
Optimization problem	which rigid bracings involve fewest braces? ■

Example 2.2: job assignment

In discussing a job assignment problem, we might wish to consider any of the following questions:

Existence problem	is it possible to assign all of the applicants to jobs?
Construction problem	if such an assignment exists, how can we construct one?
Enumeration problem	how many job assignments are there, and can we list them all?
Optimization problem	which job assignments are 'best' (according to some criterion)? ■

Problem 2.1

Classify each of the following problems as an *existence*, *construction*, *enumeration* and/or *optimization* problem.

(a) Is there any guaranteed method for finding your way out of a maze?

(b) How do you devise a suitable code to enable a spacecraft to communicate with Earth?

(c) What is the shortest route from Land's End to John O'Groats?

(d) What are the best locations for fire stations in a city?

(e) How many molecules are there with the formula C_6H_{14}?

(f) Can a floor be tiled using a combination of regular twelve-sided, six-sided and four-sided tiles?

Note that some problems may be classified under more than one heading.

We now look at each class of problem in turn.

2.2 Existence problems

Faced with any problem, it is natural to ask *does a solution exist?* The following examples, chosen from the *Graphs* part of the course show that this is not always an easy question to answer. The first example is the Königsberg bridges problem, which you met in Section 1.4.

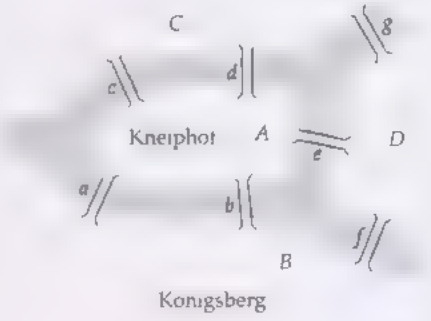
Example 2.3: Königsberg bridges

The question faced by the citizens of Königsberg was:

> is it possible to find a route that crosses each bridge exactly once and ends at the starting point?

This is clearly an example of an *existence* problem, since it is concerned with whether or not a suitable route exists.

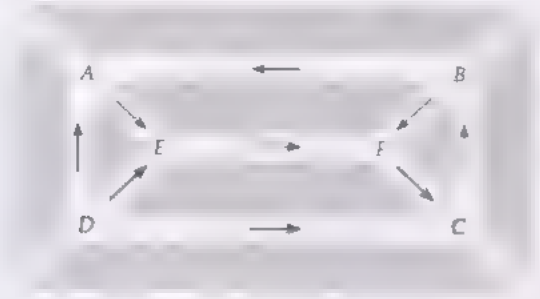
Note that, in order to demonstrate that such a route exists, it is sufficient to produce a specific route. However, to show that such a route does *not* exist, we must actually *prove* that this is the case. ■



Königsberg

Problem 2.2

Consider the following one-way system in a town:



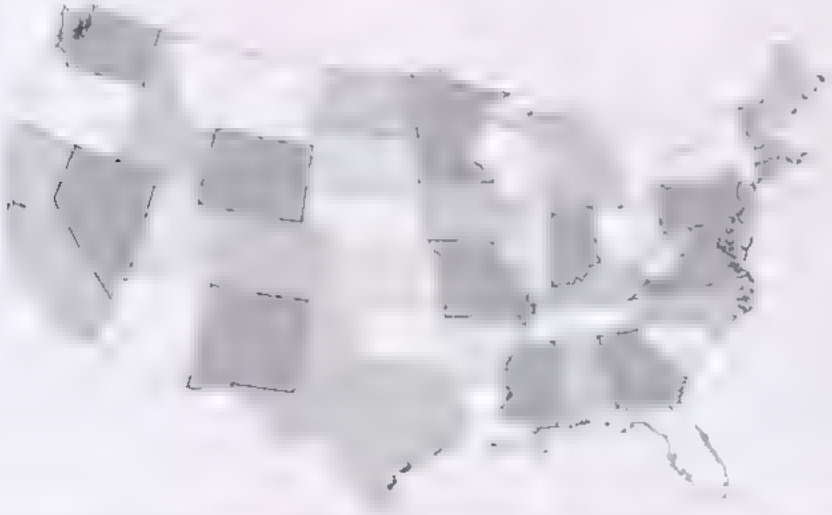
Within this one-way system, does there exist a route from:

(a) *A* to *B*? (b) *C* to *D*? (c) *F* to *E*?

In each case, either give a specific route, or explain why such a route cannot exist.

Another celebrated example of an existence problem is the map colouring problem, which you met in Section 1.1.

Example 2.4: map colouring



If you used your computer to colour the map of the USA, you should have found that it is possible to colour it with just four colours. So the existence problem

is there a four-colouring of the map of the USA?

is answered in the affirmative by finding just a single four-colouring, such as the one given above. This observation leads to the more general question:

can every map be coloured with four colours?

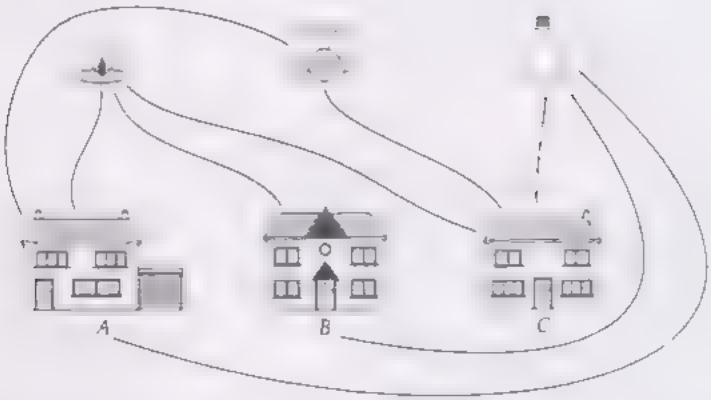
This question can also be regarded as an existence problem, since it can be reformulated as:

does there exist a map that needs more than four colours?

Note that, in order to answer this question in the affirmative, it is sufficient to produce a map that requires five colours. However, to show that *no* such map exists, it is necessary to *prove* that *every* map can be coloured with only four colours. ■

Yet another celebrated example of an existence problem is the *utilities problem*.

Example 2.5: utilities problem



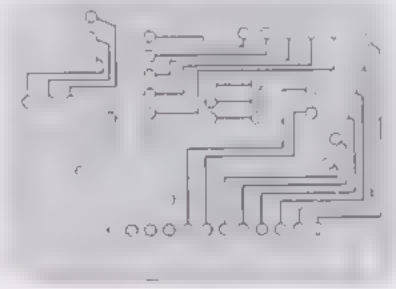
Three quarrelsome neighbours wish to connect their houses to the three 'utilities' gas, water and electricity in such a way that the various connections do not cross each other in the plane. This leads to the following existence problem:

> does there exist a way of putting in all nine connections without crossing?

In the above diagram, eight of the nine connections appear, but house B is not connected to water. It is easy to convince oneself, by trial and error, that the answer to the above existence problem is NO, but actually *proving* that no solution exists is more difficult. ■

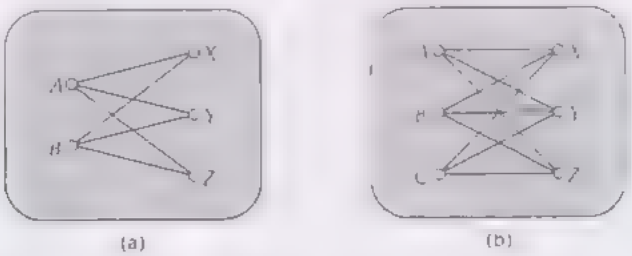
A proof is given in *Graphs 3, Planarity and colouring*.

The utilities problem is related to a number of practical problems arising in the design of printed circuit boards, on which electronic components are connected by means of conducting strips printed directly onto a flat board of insulating material. Such printed connections may not cross, since this would lead to undesirable electrical contact at crossing points.



Problem 2.3

For which of the following can you reposition the conducting strips in such a way that no crossing points occur?



We conclude this subsection on existence problems by noting that we can sometimes prove that something must exist, even though we may not know how to find a specific instance of it. For example, it is easy to see that

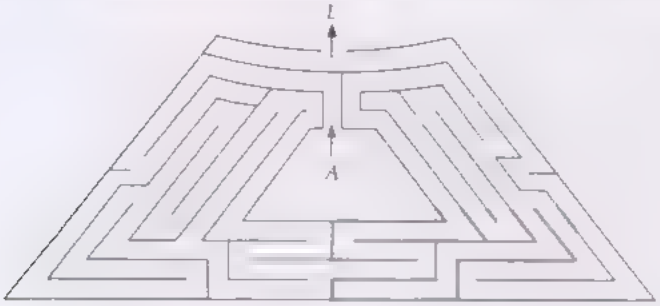
> in any group of eight people, there must exist at least two people who were born on the same day of the week,

since there are only seven days in a week and so the eight people cannot all have been born on different days. However, although we know that there *exist* at least two people who were born on the same day of the week, the above explanation does not help us to *find* who they are.

2.3 Construction problems

Once we know that a solution to a problem exists, we may wish to find a way of constructing it. As we saw above, we can often show that an existence problem has a solution by *constructing* such a solution; but this is not always the case, as we saw at the end of the previous subsection. In such cases, we know that solutions exist because there are theoretical reasons for this, but these reasons may not give us any clue as to how a specific solution may be constructed.

Problem 2.4



Use trial and error to construct a route from the centre (A) to the exit (L) of the maze at Hampton Court Palace, shown above.

Some of the simpler problems in this course, such as that in Problem 2.4, can be solved by trial and error; but many are too complex to be solved in this way, because the time involved may make trial and error impractical. For example, for a maze much more complicated than the Hampton Court maze, you would not want to use trial and error to try to find a route from the centre to the exit. Similarly, it is easy to analyse a telecommunication system that interconnects only five or six subscribers, since most problems that arise can be solved by trial and error. On the other hand, a modern telephone exchange may involve the interconnections of tens of thousands of subscribers, and so any trial-and-error approach is out of the question.

What we need for constructing solutions to complex problems is a systematic step-by-step procedure. Such a procedure is called an *algorithm*.

Historical note

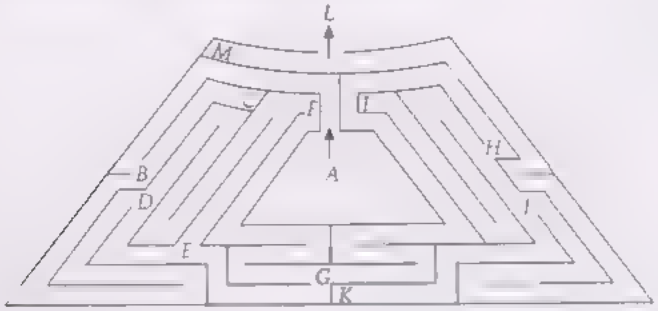
The word *algorithm* is derived from the name of the Islamic mathematician Mohammed ibn Musa Al-Khwarizmi (*c.*780–*c.*850). His text *Al-Kitab Al-Mukhtasar fi Hisab Al-Jabr W'al-Muqabala* (The Compendious Book on Calculation by Completion and Balancing) gives us the word *algebra*.

An algorithm is essentially a list of instructions to be applied, one at a time, in a given order, to appropriate input data, until a problem has been solved. You may find it helpful to think of an algorithm as similar to a recipe in a cookery book. A recipe consists of a list of ingredients, corresponding to the input data, and a list of instructions to be carried out in a particular order. To make a particular dish, you take the ingredients and follow the recipe. Similarly, a knitting pattern can be thought of as a form of algorithm, since it consists of a list of instructions that need to be carried out in a particular order.

Note that it is essential to know when we have completed all the steps in an algorithm, for otherwise we could carry on for ever. Therefore every algorithm must have a STOP instruction, to indicate when the procedure terminates.

Problem 2.5

(a) Consider the following map of the Hampton Court maze, in which each junction and dead-end is labelled.



Use these labels to construct, in a systematic manner, a route from the centre (A) to the exit (L).

(b) Use your systematic approach in part (a) to help you formulate an algorithm for finding a route out of a maze, given a map of the maze and a starting point within the maze.

The algorithm you devised in Problem 2.5 makes use of a map of the maze. It will not help you much if you are stuck in a maze without a map. However, it does help in devising an algorithm that *will* get you out. The algorithm is based on the following rule:

> never return along the passage that led you to a junction in the first place, unless there is no alternative.

By following this rule at each junction, you can escape from any maze, passing at most twice (once in each direction) along each passage.

Historical note

The above rule is due to Gaston Tarry, who published it in 1895. The main difficulty with the rule is in recognizing which of the passages leading to a junction was the one that led you there in the first place. Tarry also gave rules for doing this, based on a system of markers to be left at each junction.

Much of this course is involved with the study of algorithms for solving particular problems. In Section 4, we shall look at two algorithms for minimum connector problems, and one that provides an approximate solution to travelling salesman problems.

2.4 Enumeration problems

Once we know that a particular problem has a solution, and we know how to construct such a solution, the next questions are *how many solutions are there?* and *what are they?* For example, in Section 1.7, we constructed one solution to the given job assignment problem, and then asked you to list all the solutions.

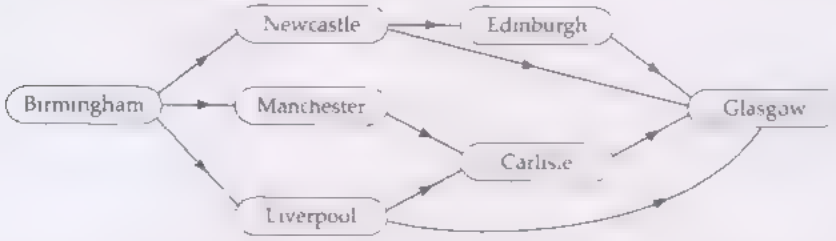
We shall distinguish between these two types of enumeration problem. A *counting problem* is one in which we wish to know *how many* objects of a certain kind there are; a *listing problem* is one in which we wish to produce a *list* of all these objects.

Rules of counting

In solving counting problems, there are some useful rules that we can use — namely, the *addition rule*, the *multiplication rule*, and the *principle of inclusion–exclusion*. The following example illustrates these rules.

Example 2.6: counting routes

In the following route map, how many different routes are there from Birmingham to Glasgow? List them.



To solve the counting problem, note that any route must go via Newcastle, Manchester or Liverpool. There are two possible routes via Newcastle, one route via Manchester, and two routes via Liverpool. The solution of the counting problem is therefore $2 + 1 + 2 = 5$.

To solve the corresponding listing problem, we must list the five solutions explicitly. These are:

Birmingham → Newcastle → Edinburgh → Glasgow

Birmingham → Newcastle → Glasgow

Birmingham → Manchester → Carlisle → Glasgow

Birmingham → Liverpool → Carlisle → Glasgow

Birmingham → Liverpool → Glasgow ■

In the above example, we split the set of all possible routes from Birmingham to Glasgow into three mutually exclusive subsets, and then added together the numbers of routes in each subset. This illustrates an important principle known as the *addition rule*.

Addition rule

The number of objects in a set can be counted by splitting the set into disjoint subsets and adding together the numbers of objects in each subset.

Two subsets are *disjoint* if they have no elements in common

Example 2.6 continued

Now suppose that, in addition to the above routes, there are three routes from Glasgow to Los Angeles: via Chicago, via New York, and via Atlanta. How many routes are there from Birmingham to Los Angeles via Glasgow?

To answer this, note that, for each of the five routes from Birmingham to Glasgow, there are three possible routes continuing from Glasgow to Los Angeles. Thus, the total number of routes from Birmingham to Los Angeles via Glasgow is $5 \times 3 = 15$. ■

This illustrates the following *multiplication rule*.

Multiplication rule

If a counting problem can be split into stages, each consisting of a number of options, then the total number of possibilities is obtained by multiplying the numbers of available options at each stage.

Example 2.6 continued

Now suppose that 82 airline passengers flew from Birmingham to Glasgow, 73 flew from Glasgow to Los Angeles, and 38 passengers were included on both of these flights. How many passengers were there altogether?

To answer this, note that, if we simply add the numbers of passengers on the two separate flights, then the 38 passengers on both flights will be counted twice. We must therefore compensate for this by subtracting them from the sum. The total number of passengers is therefore $82 + 73 - 38 = 117$. ■

This illustrates the following important principle.

Principle of inclusion–exclusion

If a set S can be split into two subsets A and B, not necessarily disjoint, then the number of elements in S is equal to

(number of elements in A) + (number of elements in B)

– (number of elements common to both A and B).

The set of elements common to both A and B is called the *intersection* of A and B and is denoted by $A \cap B$.

We illustrate these rules of counting by some more examples.

Example 2.7: binary words

A *binary word* is a string of 0s and 1s, such as 01, 1001 or 11101; a single binary digit (0 or 1) is called a *bit*. How many binary words contain exactly n bits?

To solve this counting problem, we note that there are exactly two options (0 or 1) for each bit, and so, by the multiplication rule, the total number of binary words is

$$\underbrace{2 \times 2 \times \ldots \times 2}_{(n \text{ terms})} = 2^n.$$

For a given value of n, we can list all the various possibilities. For example, if $n = 4$, then there are sixteen binary words:

0000, 0001, 0010, 0100, 1000, 0011, 0101, 0110,
1001, 1010, 1100, 0111, 1011, 1101, 1110, 1111. ■

Example 2.8: divisibility of integers

How many integers between 1 and 60 (inclusive) are divisible either by 4 or by 5? List them.

To solve the counting problem, note that between 1 and 60 there are:

$60/4 = 15$ integers divisible by 4;

$60/5 = 12$ integers divisible by 5.

However, some of these integers are divisible by both 4 and 5 (those divisible by 20); between 1 and 60 there are

$60/20 = 3$ integers divisible by both 4 and 5.

It follows from the principle of inclusion–exclusion that the answer is

(number divisible by 4) + (number divisible by 5)
– (number divisible by both 4 and 5),

which is $15 + 12 - 3 = 24$

The solution to the corresponding listing problem is

4, 5, 8, 10, 12, 15, 16, 20, 24, 25, 28, 30, 32,
35, 36, 40, 44, 45, 48, 50, 52, 55, 56, 60. ■

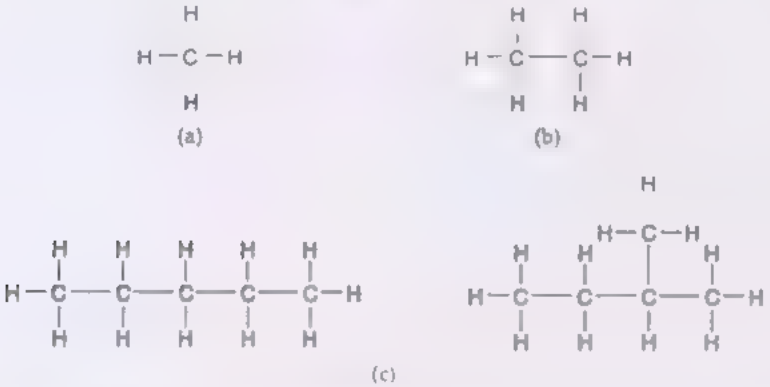
In general, problems of counting and listing may be closely related. For example, the easiest way of counting something is often to construct a list of all possibilities and then to count how many there are. Indeed, for some counting problems this may be the *only* known method of solution — for example, when counting the number of times the digit 3 appears in the first hundred significant figures of π.

In many instances the listing problem is much harder to solve than the corresponding counting problem. In fact, there are many problems for which the answer to the counting problem is known, but no one has been able or willing to list all the possibilities. An example of this follows.

Example 2.9: chemistry

A molecule consists of a number of atoms linked by chemical bonds. For example, a molecule of methane (CH_4) consists of a carbon atom (C) bonded to four hydrogen atoms (H), and may be represented by diagram (a) below. Similarly, a molecule of ethane (C_2H_6) consists of two carbon atoms bonded to six hydrogen atoms, and may be represented by diagram (b) below. More generally, an *alkane* (or *paraffin*) is a molecule with formula C_nH_{2n+2}, for some positive integer n. If an alkane has four or more carbon atoms, then there exist different molecules (known as *isomers*) with the same formula, as illustrated by diagram (c) below, for C_5H_{12}.

Each carbon atom is bonded to exactly four atoms, and each hydrogen atom is bonded to exactly one atom (a carbon atom).



(a) (b) (c)

The problem of *counting* the number of different alkanes C_nH_{2n+2} for any given value of n has been solved, although there is no simple formula for the answer. However, the problem of *listing* such molecules for a given value of n has not been solved, except when n is small. For example, the number of different molecules with the formula $C_{25}H_{52}$ is known — it is over a million — but no one has ever made a complete list of them. ■

We return to the counting of alkanes in *Graphs 1, Graphs and digraphs* and in *Graphs 2, Trees*.

Problem 2.6

Show that there is just one alkane with formula C_3H_8, but that there are two alkanes with formula C_4H_{10}.

2.5 Optimization problems

For many combinatorial problems it is not enough to know that a given problem has a solution. It may not even be enough to be able to construct a solution using an algorithm, or to count or list all the solutions. In many cases we need to find the 'best' solution, and part of the problem may be in deciding what is meant by the word 'best'.

You met several examples of optimization problems in Section 1. For example, in Section 1.8, you were asked to find the 'best' route between two stations of the London Underground. In this case, your 'best' route might be a route that involves going through the smallest number of stations on the

way, or a route that involves the smallest number of changes, or some mixture of the two. In any case, when solving such a problem, you may need to seek some further information, such as the time required to change lines, before you can determine the best solution.

In Section 1.8, you were also asked to find the shortest route (in hours of travel time) between pairs of American cities. Here, the meaning of 'best' is clear, as any route that takes the shortest travel time is a 'best' solution. However, even for this problem, you may want to take other factors into account, such as the amount of expressway (motorway) driving, the attractiveness of the scenery, and so on.

Such considerations are all part of the modelling process, which we consider in Section 5.

Another optimization problem that you met earlier was a network flow problem (Section 1.5). Here the problem is that of determining the maximum flow of fluid that can be sent from one place to another. Again, the meaning of 'best' is clear, although there may be more than one way of achieving a maximum flow. In such a case, there may be practical reasons why one solution is preferable to another.

Finally, two other types of optimization problem that you met in Section 1 are minimum connector problems (Section 1.9) and travelling salesman problems (Section 1.10). In Section 1.9 you were asked to find a minimum connector joining a number of towns, and in Section 1.10 you were asked to find a route of least total length. Again, the meaning of 'best' is clear, although there may be external considerations that indicate one particular solution as the most appropriate.

Optimization problems of this kind occur frequently throughout the course, particularly in the *Networks* units. In such problems, we usually want to maximize or minimize some given parameter (distance, flow, time, etc.), and we can sometimes do this by carrying out an appropriate algorithm. The method used in many algorithms is first to guess at a solution to the problem that satisfies all the required conditions, but is usually not the 'best' solution. We then try to improve this solution, step by step, until we finally obtain an optimum solution. As long as this final solution can be obtained in a finite (and, preferably, small) number of steps, and we can tell when we have reached the optimum solution, this method is a good one.

For many optimization problems, it is not difficult to find suitable algorithms that can be applied quickly and efficiently; examples of such problems are shortest route problems, job assignment problems and minimum connector problems. However, there are also many problems for which no efficient algorithms are known, although there may be 'heuristic' methods that work well in practice; these are methods that are quick to apply, but do not necessarily lead to a correct solution —though they usually lead, at least, to a reasonable approximation to the correct solution. Travelling salesman problems form a class of problems for which no efficient algorithms exist, but for which there are reasonably successful heuristic methods.

The efficiency of algorithms is discussed in Section 4.

You will see a heuristic approach to travelling salesman problems in Section 4, and others will be given later in the course.

After studying this section, you should be able to:

- classify combinatorial problems as *existence*, *construction*, *enumeration* or *optimization* problems, and give examples of each class;
- explain the meaning of the term *algorithm*;
- understand and use the *addition* and *multiplication rules* of counting, and the *principle of inclusion–exclusion*.

3 Representing situations diagrammatically

We have seen how various problems can be expressed in combinatorial terms — as existence, construction, enumeration or optimization problems — and such representations give us a convenient way of formulating these problems. We shall see several instances where the act of formulating a problem in precise terms is a major step towards solving it. In fact, expressing a problem in combinatorial terms often gives us a clue as to how difficult the problem is likely to be, and this information is frequently of great importance. Moreover, once a problem has been expressed in combinatorial terms, we may find that we can solve it, or parts of it, by using standard combinatorial techniques or previously tried methods. Although many problems cannot be solved in this way, we sometimes find that various parts of a problem yield to a combinatorial approach, these can be solved independently of the rest of the problem. Alternatively, we may fail to solve the problem in hand, but may develop some mathematics which is worthy of study in its own right, we sometimes find later that this mathematics is just what is needed to solve some other, apparently unrelated, problem.

3.1 Graphs and digraphs

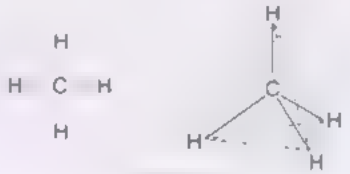
Graphs

In order to introduce the idea of a *graph*, we return to some of the problems and diagrammatic representations that we introduced earlier.

Example 3.1: London Underground

In Section 1.8 we considered the problem of finding a route between two stations on the London Underground. As we saw earlier, the Underground map does not represent every feature of the city, such as the exact geographical locations of the stations, but does represent diagrammatically the way in which the stations are interconnected. ■

Example 3.2: chemistry



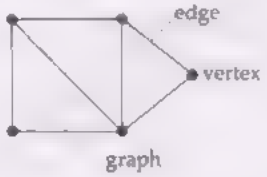
methane

In Section 2.4 we considered the problem of counting molecules of the form C_nH_{2n+2}. We saw how such a molecule can be represented diagrammatically, with atoms indicated by their chemical symbols, and chemical bonds shown by lines linking these symbols. Note that diagrams of this sort do not tell us how the atoms are aligned in space; for example, the hydrogen atoms of methane do not all lie in a plane, but are situated at the vertices of a regular tetrahedron, with the carbon atom at the centre. Nevertheless, such diagrams are extremely useful for illustrating how the various atoms are connected together, and we can obtain much information about the likely chemical behaviour of a molecule by studying its diagram. ■

Example 3.3: utilities problem

In Section 2.2 we described the utilities problem, in which three neighbours wish to be connected to three utilities: gas, water and electricity. The problem is to decide whether all nine connections can be inserted without any 'crossings'. ■

In each of these examples, we have a system of 'objects' that are interrelated in some way — stations interconnected by railway lines, atoms linked by chemical bonds, and neighbours connected to utilities. In each case, we can draw a diagram in which the objects are represented by points, and the interconnections between pairs of objects are represented by lines (not necessarily straight lines) between the corresponding points. Such a diagram is called a *graph*, the points representing the objects are called *vertices*, and the lines representing the interconnections are called *edges*.



We can express these ideas as follows.

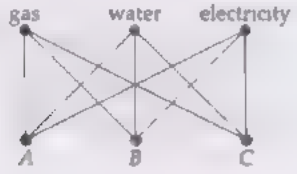
Definition

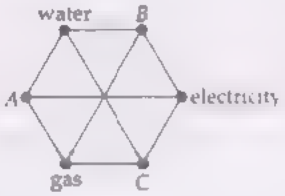
A **graph** is a diagram consisting of points, called **vertices**, joined together by lines, called **edges**; each edge joins exactly two vertices.

A slightly more formal definition is given in *Graphs 1, Graphs and digraphs*. Note that the terminology is not completely standard; some authors use *node* or *point* for what we call a vertex, and *arc* or *line* for what we call an edge.

The London Underground map is essentially a graph, in which the stations are the vertices and the railway lines are the edges, as is the diagram of a molecule, in which the atoms are the vertices and the chemical bonds are the edges.

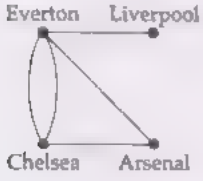
The utilities problem can be represented by a graph with six vertices, corresponding to the three neighbours and the three utilities, and nine edges, corresponding to the nine possible connections. The following diagrams illustrate two possible drawings of this graph. The utilities problem is that of finding yet another drawing that involves no crossing of edges.





Note that the three neighbours are not joined to each other, nor are the utilities.

The concept of a graph is a simple one, and graphs can be used whenever we wish to depict interconnections or relationships between objects. For example, any of the following graphs, with four vertices and five edges, can be used to depict the five football games played in a certain period between four teams — Arsenal has played once against Chelsea and Everton, but not against Liverpool; Chelsea and Everton have played each other twice; and Liverpool has played Everton once.



We can draw a graph in many ways, as long as it represents the same interconnections; each of the above drawings represents the same information about which teams have played which, and we regard them as the same graph.

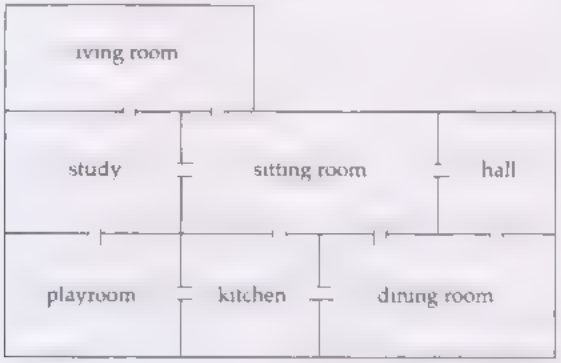
Problem 3.1

Represent each of the following situations by a graph:

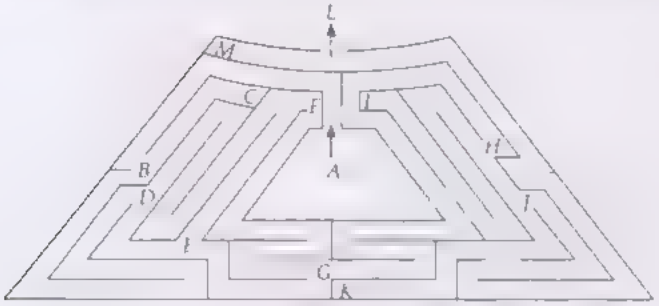
(a) the following friendships among four people:

John is friends with Joan and Jill, but not Jack;
Jack is friends with Jill, but not Joan;
Joan is friends with Jill

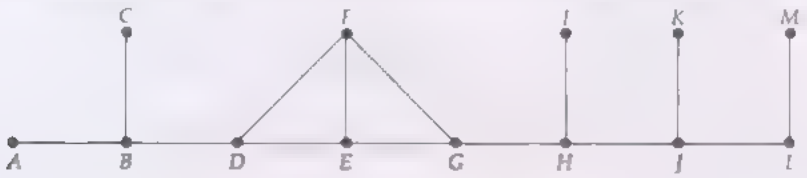
(b) the following floor plan of the ground floor of a house:



Problem 3.2



The Hampton Court maze may be represented by a graph as follows:



In this graph, a vertex represents a junction or dead-end, and two vertices are joined by an edge if there is a route between the corresponding locations that does not pass another such location.

Use this graph to list all the routes from the centre (A) to the exit (L) of the Hampton Court maze that do not involve retracing your steps.

In general, the representation of a situation by a graph is often a major step in the solution of a problem — as you saw in the case of maze tracing in Problem 3.2, and as the following problem also shows.

Problem 3.3

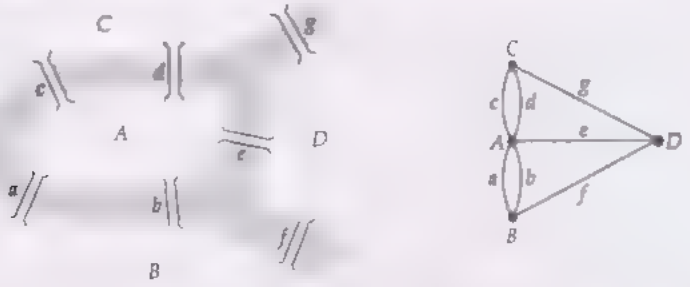
Suppose that there are six people at a party. Prove that it is always possible to find *either* three people who all know each other *or* three people none of whom knows either of the other two.

Hint Represent the six people by the vertices of a graph. Then consider the five edges (thick or thin) emerging from any one of the vertices, where two vertices are joined by a thick edge if the corresponding people know each other and by a thin edge otherwise.

We now return to some of the problems in Section 1, and see how they can be represented by graphs.

Example 3.4: Königsberg bridges

In this problem, there are four land areas interconnected by seven bridges. We can represent these interconnections by a graph with four vertices, corresponding to the four land areas, and seven edges, corresponding to the seven bridges.



The problem of crossing each of the seven bridges exactly once and returning to the starting point has now been transformed into a graph problem:

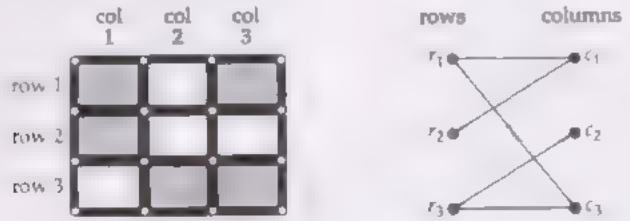
> can you draw the above graph and return to your starting point without lifting your pen from the paper and without tracing any edge twice?

■

If you wish, you can now proceed directly to the computer activities on the **Königsberg bridges revisited**.

Example 3.5: braced rectangular frameworks

In Section 1.6, you met the problem of deciding where to brace a given rectangular framework to make it rigid. Each brace occurs in one row and one column of the framework, and we can record the positions of the various braces by drawing a graph whose vertices correspond to the rows and columns of the framework and whose edges correspond to those rows and columns where a brace appears. For example, in the following framework there is a brace in row 1, column 1, but there is no brace in row 1, column 2, in the corresponding graph, there is an edge joining the row 1 vertex r_1 to the column 1 vertex c_1, but there is no edge joining the row 1 vertex r_1 to the column 2 vertex c_2.



In *Design 2, Kinematic design*, you will see that the problems of determining whether a given braced rectangular framework is rigid, and whether any braces can be removed without affecting the rigidity, can be answered directly by studying the corresponding graph. ■

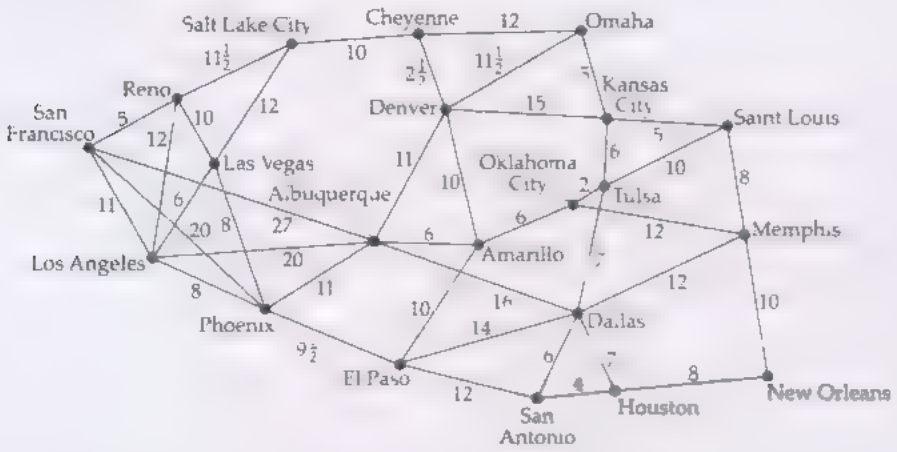
Such a graph, in which the vertices split naturally into two sets, with each edge joining vertices in different sets, is called a *bipartite graph*. In this case, the two sets correspond to the rows and the columns. Another example of a bipartite graph is the graph arising from the utilities problem, where the two sets correspond to the neighbours and the utilities. Bipartite graphs feature throughout this course.

If you wish, you can now proceed directly to the computer activities on **Braced rectangular frameworks revisited.**

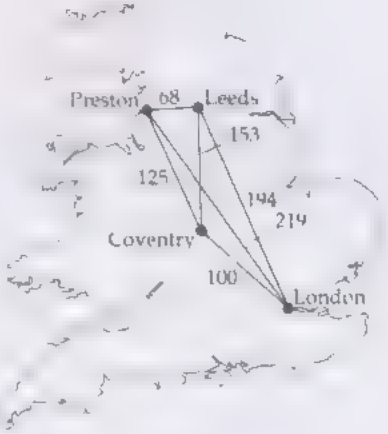
Weighted graphs

In Section 1.8, on optimal routes, you saw the following road map of part of the USA. This road map has the form of a graph, with vertices representing the cities and edges representing the roads joining them in pairs. Each edge has a number associated with it, representing the *travel*

time (in hours) between a neighbouring pair of cities. These numbers are called **weights**, and a graph with a weight associated with each edge is called a **weighted graph**.



Another example of a weighted graph appeared in Section 1.10, on travelling salesman problems. This weighted graph has four vertices, corresponding to London, Coventry, Preston and Leeds, and six edges joining them. Here, the weight on each edge represents the *distance* between the corresponding pair of cities.



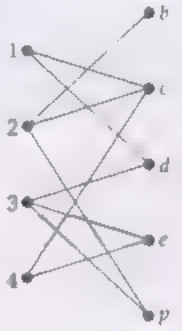
The weight on an edge can refer to many things. For example, on a road map, it may represent the *distance*, *time* or *cost* involved in travelling along the edge.

Thus is another example of a bipartite graph

Example 3.6: job assignment

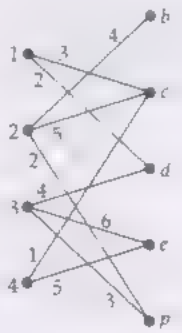
We can represent the particular job assignment problem in Section 1.7 by the following graph, in which each edge links an applicant to a job for which he or she has applied:

applicant	job
1	*c*, *d*
2	*b*, *c*, *p*
3	*d*, *e*, *p*
4	*c*, *e*



Suppose now that the applicants have different abilities to do various jobs. In this case, we can assign to each edge a number — usually referred to as a **cost** — representing the ability of the corresponding applicant to do the particular job; the lower the cost, the more able is the applicant to do that job. For example, consider the following table of costs and the corresponding weighted graph:

		job				
		b	*c*	*d*	*e*	*p*
applicant	1	–	3	2	–	–
	2	4	5	–	–	2
	3	–	–	4	6	3
	4	–	1	–	5	–



Then, the solution

1–carpenter, 2–plumber, 3–decorator, 4–electrician,

has a total cost of $3 + 2 + 4 + 5 = 14$. ■

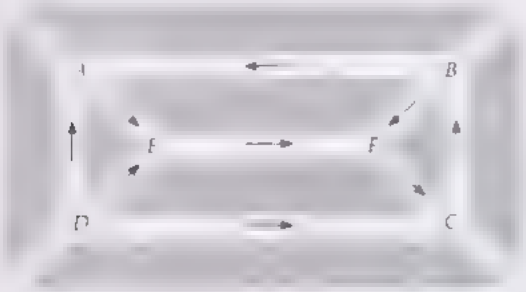
Notice that optimal route problems, travelling salesman problems and job assignment problems, when considered as problems involving weighted graphs, involve trying to find a set of edges of minimum weight. They are therefore examples of ***minimum-weight problems*** (or ***minimum-cost problems*** in those cases where the weights are referred to as costs). Many of the optimization problems you will meet in the course can be formulated as minimum-weight problems.

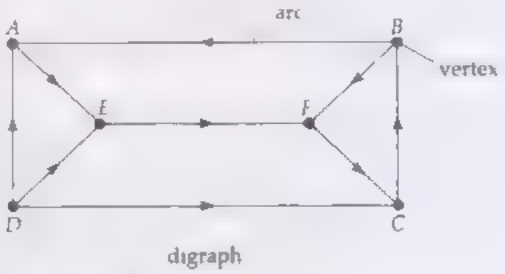
Digraphs

Consider again the one-way street system introduced in Problem 2.2.

Because the streets are all one-way, we cannot faithfully represent this system by a graph, because the edges of a graph are *undirected lines*, but we can represent it by a similar diagram in which we put arrows on the edges to indicate the directions of the one-way streets, creating *directed lines* called *arcs*. Such a diagram is called a *digraph*.

The word *digraph* is an abbreviation of *directed graph*.





digraph

Definition

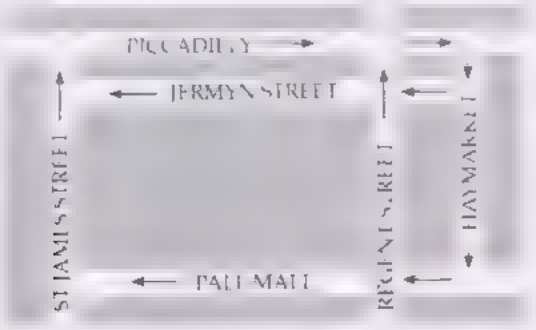
A digraph is a diagram consisting of points, called vertices, joined together by directed lines, called arcs; each arc joins exactly two vertices.

A slightly more formal definition is given in *Graphs 1, Graphs and digraphs*.

Another example of a digraph is the *Plan of the unit* diagram on page 2, each vertex corresponds to a section of the unit, and the arcs can be used to trace possible study routes through the unit.

Problem 3.4

Represent the following one-way street system by a digraph:



Just as we can assign weights to the edges of a graph to form a weighted graph, so we can assign weights to the arcs of a digraph to form a **weighted digraph**. An example of a weighted digraph is the pipeline network in Section 1.5.

3.2 Networks

The word *network* is one that commonly arises in everyday life. We speak of a *rail network* as a system of stations interlinked by railway lines, or a *road network* as a system of towns interlinked by roads. We speak of a programme produced by a television company being *networked* around the country. We speak of an *electrical network* involving terminals interconnected by wires, or a *telecommunication network* involving interlinked telephone exchanges and subscribers.

In this course, we shall also use the word *network* in a more specialized sense. A graph or digraph represents only the *structure* of a system; a **network** on the other hand, is a graph or digraph that carries some additional numerical information. This information depends on the particular application under consideration, but may consist of weights associated with the edges or arcs. However, a network is not necessarily just a weighted graph or digraph, for example, it may possess two sets of weights on the edges or arcs — representing capacities and costs, say — and there may also be weights associated with the vertices.

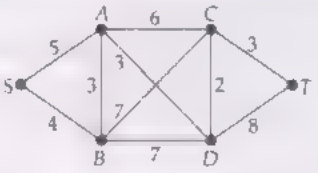
Example 3.7: road networks

In a road network, the weight on each edge or arc (road) may correspond to one of the following: its length in miles or kilometres, the estimated or actual time taken to travel along it; the cost involved in travelling along it (fuel, tolls, etc.).

Now suppose that we wish to find the route of shortest length, using trunk roads, between two particular towns. In this problem we are not concerned with the amount of traffic that each road can take; instead we are concerned only with the length of each section of the road, so we assign the length as the weight on the corresponding edge of our network. The problem of finding the shortest route between any two towns on our map is thus reduced to that of finding the shortest route between the two corresponding vertices on our network — that is, the route with the minimum total weight. ■

Problem 3.5

Find the shortest route from S to T in the following network.

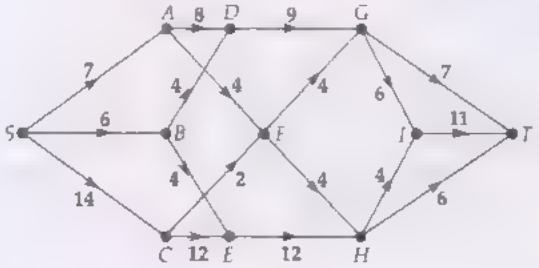


A simple shortest route problem like this one may be solved by trial and error, but for larger problems we need an algorithm. We describe such an algorithm in *Networks 2, Optimal paths*.

Hint First find the shortest route from A to T, then from B to T, then from S to T.

Example 3.8: pipeline networks

Another example of a network is the pipeline network that you met in Section 1.5.



In this network, the weight on each arc is the *capacity* of the arc, namely, the maximum amount of fluid that can be sent along that arc at any one time; for example, the arc SC has capacity 14 and the arc CE has capacity 12. ■

Note that a label on an arc representing a flow is *not* a weight in this context: a weight is a number, such as a capacity, which is part of the numerical information *carried by* the network; a flow is a number *we assign* to an arc, and so is not a weight.

Example 3.9: telecommunication networks

In Section 1.3 we considered a telecommunication system represented by a graph whose vertices correspond to exchanges and whose edges correspond to links between these exchanges.

There, we were interested in a problem about *connections* — that is, a *graph* problem. However, suppose we wish to determine the maximum number of calls that can be handled simultaneously by any part of the system. In practice, the total number of calls that can be carried between any two parts of such a system is limited. Any exchange (vertex) or link (edge) has a *capacity*, which is the maximum number of calls it can carry at any given time. Thus the problem of finding the maximum number of simultaneous calls between any two parts of the system is a *network* problem rather than a graph problem. ■

Other important types of network that you will meet later in the course are *activity networks*, for scheduling industrial projects, and *transportation networks*.

Scheduling problems are discussed in *Networks 2, Optimal paths*. Transportation problems are analysed in *Networks 3, Assignment and transportation*.

Problem 3.6

How would you represent each of the following as a network? What could correspond to the weight(s) on each edge or arc?

(a) a railway network;

(b) an airline network.

If you have not already done so, you should now carry out the computer activities for this section.

After studying this section, you should be able to:

- understand what are meant by the terms *graph, weighted graph, digraph, weighted digraph, network, vertex, edge, arc, weight* and *cost*;
- represent a simple situation by a graph, digraph or network, as appropriate.

4 Algorithms

In this section we return to the subject of algorithms. Recall that an algorithm is a systematic step-by-step procedure for solving problems. We may define an algorithm more formally as follows.

Definition

An **algorithm** is a systematic step-by-step procedure consisting of:

- a description of appropriate input data;
- a finite, ordered list of instructions, to be carried out one at a time;
- a STOP instruction, to indicate when the procedure is complete;
- a description of appropriate output data.

Often, if the input and/or output data are obvious, we shall not state them explicitly in our description of an algorithm.

The form in which an algorithm is presented varies from problem to problem, and may consist of instructions written in English or in a computer language, or in the form of a flow chart.

In many cases we can apply algorithms without the aid of a computer. On the other hand, many practical problems are far too large or complex to be dealt with in this way. For example, many problems in industry or commerce involve graphs or networks with hundreds or thousands of vertices. In such circumstances, it is necessary to express the algorithm in a form that can be implemented on a computer. This means that the various instructions must be precisely and unambiguously stated, and that the algorithm must terminate after a finite number of steps.

In this section, we present some examples of algorithms, and then discuss their efficiency.

4.1 Examples of algorithms

In this subsection, we consider two types of problem: minimum connector problems and travelling salesman problems. Minimum connector problems can be solved efficiently by means of an algorithm, and we give two algorithms that do this. We also present a 'heuristic' algorithm that gives an approximate solution to travelling salesman problems.

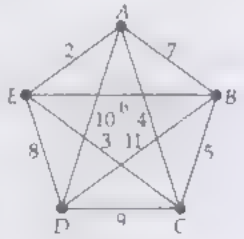
Minimum connector problems

We describe two algorithms for finding a minimum connector. Both are examples of *greedy algorithms*, so called because at each stage we are 'greedy' and choose the edge of least weight. We apply both to the following example.

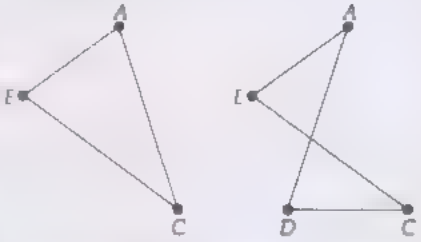
We discuss greedy algorithms again in *Graphs 2, Trees*, where we shall prove that each of the greedy algorithms given here *always* gives a minimum connector.

Example 4.1: minimum connector

Suppose that we wish to find a minimum connector linking five cities, *A*, *B*, *C*, *D* and *E*. The distances between them (in hundreds of miles) are given by the weighted graph in the margin. ■



First, we describe *Kruskal's greedy algorithm*. In applying this algorithm, at each stage we choose the edge of least weight, provided that it does not create a *cycle* — that is, a closed loop such as *ACEA* or *ADCEA* in the example above. We need to avoid cycles because, if there is a cycle, then we can decrease the total weight of our 'connector', without jeopardizing its 'connectedness', by removing any one of the edges in such a cycle.



Kruskal's greedy algorithm

START with a finite set of vertices, where each pair of vertices is joined by a weighted edge.

STEP 1 List the weights in ascending order.

STEP 2 Draw the vertices and weighted edge corresponding to the first weight in the list, provided that, in doing so, no cycle is formed. Delete the weight from the list.

Repeat Step 2 until all the vertices are connected, then STOP.

The weighted graph obtained is a minimum connector, and the sum of the weights on its edges is the total weight of the minimum connector.

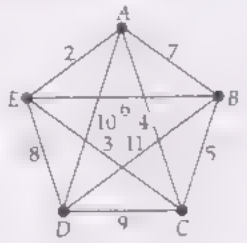
This algorithm was devised by Joseph Kruskal in 1956

When two or more weights are the same, they may be listed in any order.

If a cycle is formed, add *no* edge; simply delete the weight from the list.

Example 4.1: minimum connector by Kruskal's greedy algorithm

Consider the following weighted graph:

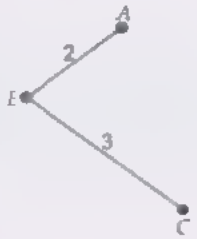


STEP 1 We list the weights in ascending order, as follows:

edge	*AE*	*CE*	*AC*	*BC*	*BE*	*AB*	*DE*	*CD*	*AD*	*BD*
weight	2	3	4	5	6	7	8	9	10	11

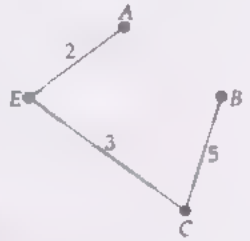
STEP 2 The first weight in the list is 2, corresponding to the edge *AE*. We draw this edge and delete 2 from the list.

STEP 2 The next weight in the list is 3, corresponding to the edge *CE*. We draw this edge and delete 3 from the list.



STEP 2 The next weight in the list is 4, corresponding to the edge *AC*. But the edge *AC* would create a cycle, so we just delete 4 from the list.

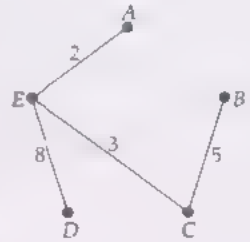
STEP 2 The next weight in the list is 5, corresponding to the edge *BC*. We draw this edge and delete 5 from the list.



STEP 2 The next weight in the list is 6, corresponding to the edge *BE*. But the edge *BE* would create a cycle, so we just delete 6 from the list.

STEP 2 The next weight in the list is 7, corresponding to the edge *AB*. But the edge *AB* would create a cycle, so we just delete 7 from the list.

STEP 2 The next weight in the list is 8, corresponding to the edge *DE*. We draw this edge and delete 8 from the list.



All the vertices are now connected, so we STOP.

Thus we obtain a minimum connector with total weight

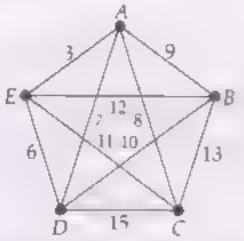
$$2 + 3 + 5 + 8 = 18.$$ ■

It can be shown that Kruskal's greedy algorithm *always* gives a minimum connector

We explain why this simple algorithm always gives a minimum connector in *Graphs 2, Trees*

Problem 4.1

By using Kruskal's greedy algorithm, find a solution to the minimum connector problem in Section 1.9.



We can summarize Kruskal s greedy algorithm by a single rule, as follows

Summary of Kruskal's greedy algorithm

To construct a minimum connector, build it up step by step by successively adding an edge of minimum weight in such a way that no cycle is created.

Although Kruskal's greedy algorithm can be applied easily by hand when the number of vertices is small, it is not so good for large problems This is because such problems would take too long to do by hand and because Kruskal's greedy algorithm is not particularly well suited to *efficient* computer implementation, due to the need to arrange the edges in order of ascending weight and the need to recognize cycles as they are created

Both of these difficulties are easily overcome by a slight modification of the algorithm; the result is called ***Prim's greedy algorithm***. In applying this algorithm, we start with *any* vertex and build up the solution step by step from there.

Prim's greedy algorithm

This algorithm was devised by R. C. Prim in 1957

START with a finite set of vertices, where each pair of vertices is joined by a weighted edge.

STEP 1 Choose and draw any vertex.

STEP 2 Find the edge of least weight joining a drawn vertex to one *not* currently drawn. Draw this weighted edge and the corresponding new vertex.

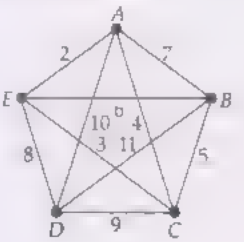
If there are two or more such edges, choose any one of them.

Repeat Step 2 until all the vertices are connected, then STOP.

The weighted graph obtained is a minimum connector, and the sum of the weights on its edges is the total weight of the minimum connector.

Example 4.1: minimum connector by Prim's greedy algorithm

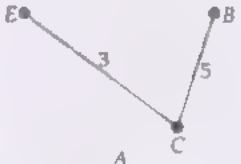
Consider the following weighted graph:



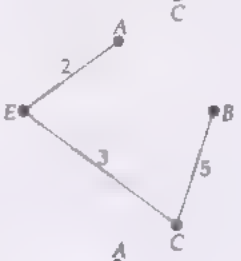
STEP 1 We choose and draw any vertex, B say.

STEP 2 The edge of least weight emerging from B is BC, with weight 5. We draw this edge.

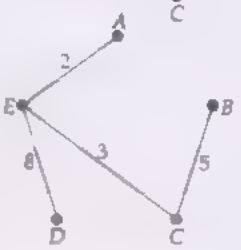
STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is CE, with weight 3. We draw this edge.



STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is AE, with weight 2. We draw this edge.



STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is DE, with weight 8. We draw this edge.



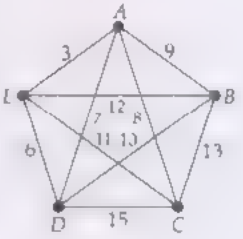
All the vertices are now connected, so we STOP.

Thus we obtain a minimum connector with total weight

$$5 + 3 + 2 + 8 = 18.$$ ■

Problem 4.2

By using Prim's greedy algorithm, find a solution to the minimum connector problem in Section 1.9, starting with vertex B.



We can summarize Prim's greedy algorithm by a single rule, as follows.

Summary of Prim's greedy algorithm

To construct a minimum connector, choose any vertex and then build it up step by step by successively adding an edge of minimum weight joining a vertex currently included to one not currently included.

Travelling salesman problems

We conclude this subsection by describing a *heuristic* algorithm for travelling salesman problems — a **heuristic algorithm** is an algorithm that does not necessarily give a correct answer, but at least gives a good approximation to one. Unlike the above algorithms for minimum connector problems, which *always* provide a correct answer, no *efficient* algorithm is known that always provides a correct answer to any given travelling salesman problem. However, the following heuristic algorithm always yields a suitable route, whose length is at most twice that of a shortest such route.

Other algorithms for travelling salesman problems are considered in *Graphs 2, Trees* and *Graphs 4, Graphs and computing*.

Thus if the length of a shortest such route is 1000, then this algorithm yields a route whose length lies in the range 1000 to 2000.

The idea of the algorithm is to build up the required route step by step, starting with a single vertex. It is very similar to Prim's greedy algorithm, except that we try to build up a *minimum cycle* (that is, a cycle of minimum weight) rather than a minimum connector.

Remember that a *cycle* is a closed loop of vertices and edges.

Heuristic algorithm for travelling salesman problems

START with a finite set of vertices, where each pair of vertices is joined by a weighted edge.

STEP 1 Choose any vertex and find the edge of least weight emerging from it. Draw the corresponding two vertices and join them by two edges, to form a cycle.

If there are two or more such edges, choose any one of them.

STEP 2 Find the edge e of least weight joining a drawn vertex v to a vertex w *not* currently drawn. Draw the new vertex w on the edge of the cycle immediately after v, moving in a clockwise direction.

If there are two or more such edges, choose any one of them.

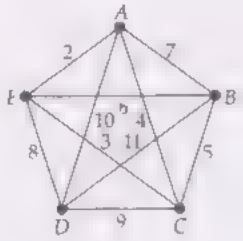
we assume that the triangle of vertices is strongly, that is, there is no triple of vertices such that the weight of edge AC is greater than the sum of the weights AB and BC.

Repeat Step 2 until all the vertices appear in the cycle, then assign the appropriate weight to each edge and STOP.

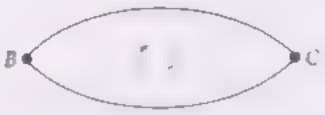
The weighted cycle obtained is a cycle through all the vertices, and its total weight — given by the sum of the weights on its edges — is at most twice the weight of the required minimum cycle.

Example 4.2: travelling salesman problem by a heuristic algorithm

We wish to find a minimum cycle in the following weighted graph:

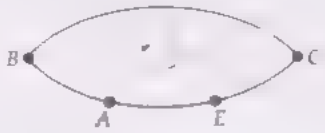


STEP 1 We choose any vertex, B say. The edge of least weight emerging from B is BC, with weight 5. We draw B and C and join them by two edges, to form a cycle.



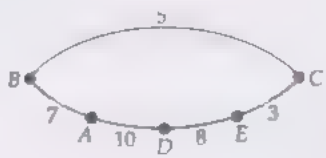
STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is CE, with weight 3. We draw the vertex E after C in the cycle.

STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is AE, with weight 2. We draw the vertex A after E in the cycle.



STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is DE, with weight 8. We draw the vertex D after E in the cycle.

All the vertices now appear in the cycle. We assign the appropriate weight to each edge and STOP.



Thus we obtain the cycle $BCEDAB$ with total weight

$$5 + 3 + 8 + 10 + 7 = 33.$$

■

Note that this cycle is not a minimum cycle. For example, the cycle $ACBDEA$ has total weight 30.

Problem 4.3

Apply the heuristic algorithm to Example 4.2, but this time starting with vertex A.

4.2 Efficiency of algorithms

Once an algorithm for solving a given type of problem has been constructed, a number of questions arise:

- how long will it take a computer to solve a particular problem of the given type using the algorithm?
- can moderately large problems of the given type be solved in a reasonable time?
- can we construct another algorithm that will solve problems of the given type more quickly?

Don't worry if you find this subsection difficult. We have included it here because we wish to comment on the efficiency of various algorithms as we progress through the course. A fuller discussion of the efficiency of algorithms is given in *Graphs 4, Graphs and computing*

All these questions relate to the amount of time a computer takes to solve a particular problem using the algorithm. The *efficiency* of an algorithm is a measure of the time it takes to solve a problem.

So, how do we measure the efficiency of a given algorithm? Consider the following example.

Example 4.3: job assignment

Five workers are available to carry out five tasks in a project. They are not equally good at each task. What is the 'best' assignment of workers to tasks? ■

In tackling this job assignment problem, we may start by assigning a weight, or cost, to each possible assignment of a worker to a task, where a low cost corresponds to a high ability, the optimization problem is then to find the assignment with the lowest total cost.

One way to do this is to look at *all* possible assignments, and then choose the one with the lowest total cost. For five workers and five tasks, this involves looking at

$$5 \times 4 \times 3 \times 2 \times 1 = 5! = 120 \text{ possible assignments,}$$

$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1.$

since we can assign the first worker to one of the five tasks in five ways, the second worker to one of the remaining four tasks in four ways, and so on. And to do this without a computer would not take very long. But what happens if we have a larger number of workers and tasks?

The following table shows how the number of possible assignments of n workers to n tasks increases rapidly with the number of workers.

workers (n)	5	10	15	20
possible assignments ($n!$)	120	3628 800	1.3×10^{12}	2.4×10^{18}

This rapid increase in the number of possibilities is another instance of the *combinatorial explosion*, which you met in Section 1.10, in connection with travelling salesman problems.

Thus, even with just ten workers, the problem is far too large to solve by this method without a computer. For ten workers, a computer examining a thousand possibilities per second could sort through the possible assignments in just over an hour; but, for a similar problem with twenty workers, the same computer would take over seventy-seven million years!

We shall refer to the above method, involving the examination of all possibilities, as the *exhaustion algorithm*. The time it takes is proportional to $n!$, and so we can take $n!$ to be the measure of the efficiency of the algorithm. However, because $n!$ increases rapidly as n increases, we need

You'd certainly be exhausted if you tried to use it, without a computer, for problems where $n > 5$.

to look for a more efficient algorithm — that is, an algorithm for which the time taken increases less rapidly as n increases. Such an algorithm is the *Hungarian algorithm*, where the time taken to reach a solution is proportional to n^2. If we assume that, for *five* workers and tasks, the Hungarian algorithm takes the same time as the exhaustion algorithm, then we obtain the following table comparing the approximate times for the two algorithms:

The Hungarian algorithm is discussed in *Networks 3, Assignment and transportation.*

workers (n)	5	10	15	20
exhaustion algorithm	0.12 secs	1 hour	46 years	7.7×10^7 years
Hungarian algorithm	0.12 secs	0.48 secs.	1.1 secs	1.9 secs

We are still assuming that our computer works through one thousand possibilities per second.

The differences between these times for larger values of n demonstrate the importance of having an efficient algorithm.

For some types of problem, we can find an algorithm where the time taken is proportional to n, or to n^2, or to n^3, or more generally to n^k, for some fixed number $k > 1$, or more generally still to some function $f(n)$, where there is some fixed number $k > 1$ such that $f(n) \leq n^k$ for all $n \geq N$ for some fixed number N; in each case, n is some parameter associated with the type of problem, such as the number of workers or the number of vertices or edges of a graph. Such algorithms are called **polynomial-time algorithms**, and are generally considered to be *efficient* algorithms. Problems for which polynomial-time algorithms exist are said to be **tractable**. Problems that are not tractable are said to be **intractable**.

Examples of tractable problems are maze-tracing problems, minimum connector problems and job assignment problems. A polynomial-time algorithm for maze-tracing problems is Tarry's algorithm, for which the time taken is proportional to m, where m is the number of edges in the graph of the maze. A polynomial-time algorithm for minimum connector problems is Prim's greedy algorithm, for which the time taken is proportional to n^2, where n is the number of vertices in the graph. A polynomial-time algorithm for job assignment problems is the Hungarian algorithm, for which the time taken is proportional to n^2, where n is the number of workers.

For certain other types of problem, the only known algorithms take a time proportional to the nth power of some fixed number $k > 1$ — such as 2^n, 3^n, etc. — or to some function $f(n)$, where there is some fixed number $k > 1$ such that $f(n) \geq k^n$ for all $n \geq N$ for some fixed number N; again, in each case, n is some parameter associated with the type of problem. Such algorithms are called **exponential-time algorithms**, and are generally considered to be *inefficient*.

To see why such algorithms are generally considered to be inefficient — and, hence, of little practical use — consider the following table, which compares the approximate times taken by two polynomial-time algorithms (with times n^2 and n^3) and two exponential-time algorithms (with times 2^n and 3^n), used on a computer performing a thousand operations per second.

		$n = 10$	$n = 50$
polynomial time	n^2	0.1 seconds	2.5 seconds
	n^3	1 second	2 minutes
exponential time	2^n	1 second	35700 years
	3^n	1 minute	2.3×10^{13} years

So, when $n = 50$, the two exponential-time algorithms are of no practical use at all

Problem 4.4

Calculate the corresponding times for $n = 10$ when the algorithm times are n^5 and 5^n.

Although polynomial-time algorithms are generally considered to be efficient and exponential-time algorithms are generally considered to be inefficient, some caution must be exercised in applying this general conclusion to a particular problem. For example, it is possible for the time taken by a polynomial-time algorithm to involve a large coefficient of proportionality (such as in $10^{100}n$) or a very large exponent (such as in n^{100}), so that the algorithm is not efficient in practice. However, such cases are rare, and most polynomial-time algorithms have a reasonably small coefficient and an exponent not greater than 2 or 3, and hence are efficient in practice. Furthermore, some exponential-time algorithms can be quite efficient in practice provided that n is not too large. However, such cases are also rare, and most exponential-time algorithms are inefficient for all but the smallest values of n.

Once we have found a polynomial-time algorithm for a problem, then we know that the problem is tractable. On the other hand, to show that a problem is intractable, we need to *prove* that *no* polynomial-time algorithm for it can exist. For example, it has been proved that no *general* algorithm (polynomial-time or otherwise) can be found to determine whether *any* given polygon can be used to tile an infinite flat surface without gaps or overlaps, so this problem is intractable. However, in general it is very difficult to prove that no polynomial-time algorithm exists for a given problem, and so there are many problems for which it is not known whether they are tractable or intractable.

Travelling salesman problems provide a well-known example of problems of this type. The only known algorithm that is guaranteed to solve any given travelling salesman problem is the exhaustion algorithm, in which all possible routes are examined. As we mentioned earlier, the time taken by the exhaustion algorithm is proportional to $n!$, where in this case $n + 1$ is the number of cities. Since $n! \geq 2^n$, for $n > 4$, the exhaustion algorithm for travelling salesman problems is an exponential-time algorithm. So, the only known algorithm for solving travelling salesman problems is exponential-time, but no one has yet proved that no polynomial-time algorithm for such problems can exist.

Recall that heuristic algorithms generally provide *approximate* and not exact solutions. Therefore, although the heuristic algorithm for travelling salesman problems discussed in Section 4.1 is a polynomial-time algorithm, it is *not* a polynomial-time algorithm for *solving* travelling salesman problems; rather, it is a polynomial-time algorithm for *finding approximate solutions to* travelling salesman problems.

Travelling salesman problems belong to an important class of problems called **NP-complete problems**. The types of problem in this class have the following interesting properties:

- no polynomial-time algorithm has yet been found for any of them;
- no one has proved that polynomial-time algorithms do not exist for any of them;
- it has been shown that, if a polynomial-time algorithm could be found for any *one* of them, then polynomial-time algorithms must exist for *all* of them;
- it has been shown that, if it could be proved that no polynomial-time algorithm exists for *one* of them, then polynomial-time algorithms would not exist for *any* of them.

It is currently believed that no polynomial-time algorithm exists for any of the NP-complete problems — that is, that NP-complete problems are intractable.

You will meet several other types of NP-complete problem later in the course

After working through this section, you should be able to:

- explain what is meant by an *algorithm*;
- use *Kruskal's greedy algorithm* and *Prim's greedy algorithm* to solve minimum connector problems;
- use a heuristic algorithm to find approximate solutions to travelling salesman problems;
- explain what is meant by the *efficiency* of an algorithm;
- understand what are meant by a *heuristic algorithm*, a *polynomial-time algorithm*, an *exponential-time algorithm*, a *tractable problem* and an *intractable problem*.

5 Mathematical modelling

In this section we explain what we mean by a *mathematical model* and by the *modelling process*.

5.1 Introduction to modelling

To introduce these ideas, we begin by reconsidering the Konigsberg bridges problem.

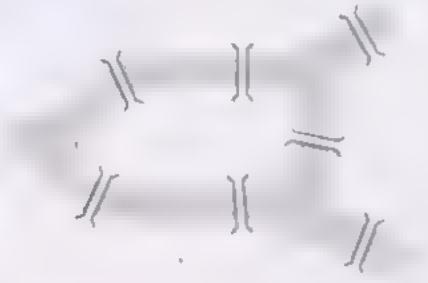
The description here essentially summarizes our approach to the problem earlier in the unit.

The citizens of Konigsberg had a practical problem concerning their Sunday walks, which they formulated in ordinary language:

> is it possible to find a route that crosses each bridge exactly once and ends at the starting point?

Tired of walking and being late for their afternoon tea, they may well have tried to solve the problem by using a small-scale three-dimensional model of the city. Such a model faithfully represents the real situation and is easier to study, but it is unwieldy and is not good for tracing routes.

A more appropriate representation is a map of the area; this represents fewer — but sufficient — features of the real situation, and is handier for tracing routes.



However, although tracing routes on this map may convince us that no such route exists, it does not help us to *prove* this.

In order to prove it, we need to identify the important features of the problem, namely the *objects* and the *ways in which they are related*:

- there are two types of object — *land areas* and *bridges*.
- each bridge connects two land areas.

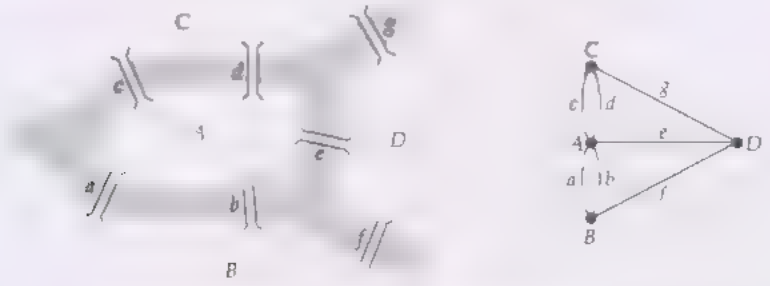
Our next step is to introduce some notation for the two types of object: we denote the land areas by the letters A, B, C and D, and the bridges by a, b,

c, d, e, f and g. We can then write the relationships between them in the form

> a connects A and B,

and so on.

This leads us to represent the situation by a *graph*, in which the vertices represent land areas and the edges represent bridges:

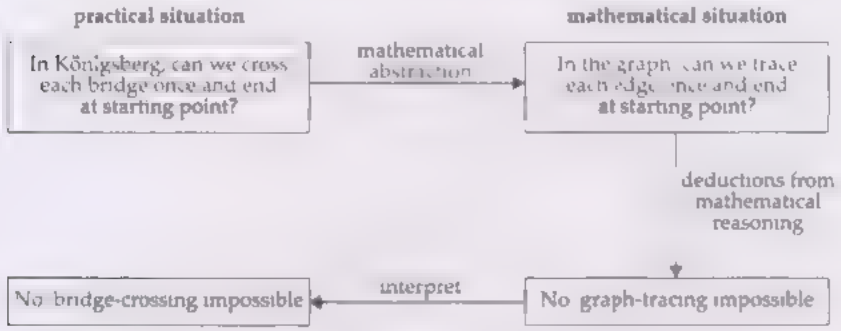


From the graph, it becomes clear that a solution can exist only if, whenever we approach a vertex along one edge, it is possible to leave that vertex by another edge: that is, *the number of edges meeting at each vertex must be even*. We see from the above graph that *none* of the vertices has this property, and thus a solution cannot exist.

The final stage is to relate the result about the graph back to the real situation:

> a route that crosses each bridge exactly once and ends at the starting point does not exist.

We can summarize the above stages by the following diagram:



This is a very simple example of *mathematical modelling*.

Problem 5.1

Describe the corresponding stages in the problem of determining whether a given braced rectangular framework is rigid, and summarize them on a diagram similar to the one above.

The modelling process that we illustrate and describe in the following two subsections is a generalization of the above ideas to more complex situations.

5.2 Location problems

This subsection comprises the television programme related to this unit, on the locating of fire stations in Rotterdam. The programme is described in the related *Television Notes*.

5.3 The modelling process

Many practical problems arising in industry, science or commerce can be formulated as mathematical problems. Such problems can then be tackled by mathematical methods, but it is important to keep in mind the relationship between the original practical problem and its mathematical formulation.

To see what is involved, we divide the modelling process into a number of stages.

1 *Identify the practical problem.*

2 *Describe the problem carefully in ordinary language.*

At this stage, interpretation, judgement and opinion are involved, so the description may well be incomplete and may not be a true reflection of the original problem.

3 *Formulate the description as a mathematical problem.*

This stage often involves simplifications that may or may not be justified. Sometimes, only one aspect of the practical problem can be represented mathematically.

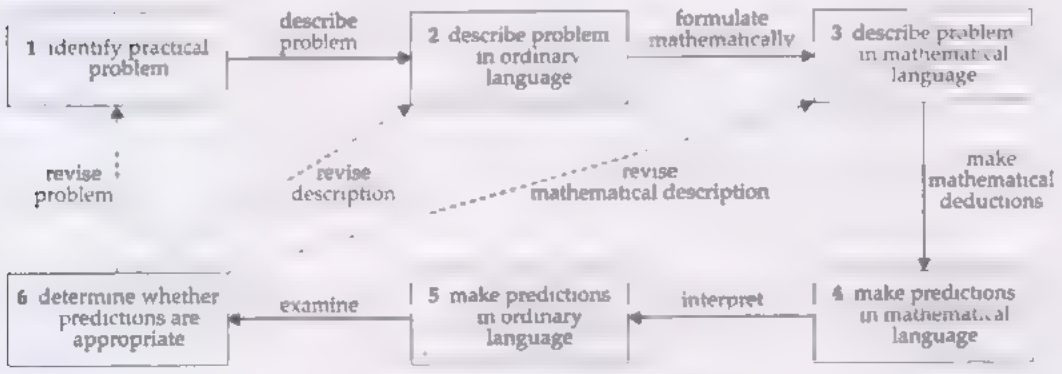
4 *Analyse and try to solve the mathematical problem.*

The mathematical problem may be solvable, or it may prove too difficult to solve. Consequently, further simplifications may need to be made before it yields to existing mathematical techniques. Alternatively, an approximate solution to the problem may be sought, rather than an exact one.

5 *Interpret the mathematical solution in terms of the original problem.*

6 *Examine the proposed practical solution to determine whether it is satisfactory.*

The simplified or approximate mathematical solution may be good enough to yield a satisfactory practical solution. On the other hand, there may have been so many simplifications or approximations that the proposed solution is of no direct practical use.



mathematical modelling process

If the conclusions are not satisfactory, it may be necessary to work through some or all of the above stages several times before an acceptable practical solution is reached. This process often involves a dialogue between a technologist and a mathematician, with each making suggestions to be adopted by the other. The mathematician may have to learn enough about the practical problem to be able to appreciate the difficulties involved in formulating it as a mathematical problem, and the technologist may have to learn enough mathematics to be able to appreciate the difficulties involved in solving the resulting mathematical problem and in interpreting the results.

The mathematical formulation of a practical problem is called a **mathematical model**. The complete process described above is called the **mathematical modelling process**.

In everyday language, a *model* is simply an abstraction of a real or imagined system. For example, a model railway incorporates many of the features of a real railway and an architect's model of a building represents many important features of the building. But any model presents a simplified view of the situation it represents, and whether these simplifications are appropriate will depend partly on the uses to which the model is to be put. If the model is 'good', then its behaviour can be directly related to the real system. But how do we determine whether a model is good?

For example, a model railway may well be acceptable to a child as a representation of the real thing, but such a model would probably be unsuitable for solving complicated problems on the movement of rolling stock around the country. Similarly, an architect's mock-up of a block of flats may be accurate enough for a town council to decide whether to start construction, but it will not contain many features of importance to the people who will eventually be living there.

An important feature of a model is that the important *components* of the real system should be *represented* by parts of the model. In a *physical model*, each part may represent one of the physical components of the real system; it may even 'look like' the component of the real system it represents, may be made of the same material, and may have the same kinematic and dynamic properties. In a *conceptual model* of the type considered in this course, the components of the real system may be represented by, for example, the vertices of a graph, digraph or network.

In addition to representing the important components of a real system, models also must represent the important *relationships* between them. For example, relationships may be represented by the edges of a graph, or by the arcs of a digraph or network.

By the time you have finished this course, you should be convinced that graphs and networks provide an excellent way of representing relationships. For this reason, they can form the basis of models of very complex systems, and provide one of the few workable ways of managing such complexity.

However, in view of our earlier comments, we must clearly be cautious when interpreting a mathematical solution in practical terms. The study of graphs and networks is a powerful tool that can give insight into many complex problems, but the conclusions drawn from it may need to be modified by additional practical considerations.

Finally, note that most problems arising in technology and science are far too complex and difficult to be solved by the methods of this course alone. Along with other mathematical tools, graphs and networks have proved their worth in many problems that face us, but they can provide only one method of approach. They can help to give us insight and understanding into these problems, but it is only in conjunction with other lines of approach that we are likely to find satisfactory solutions.

> Grant me
> The serenity to accept the problems that I cannot solve
> The persistence to solve the problems that I can
> And the wisdom to tell the difference.

Appendix: Euler's paper

The solution of a problem relating to the geometry of position

This is a translation of part of a paper of 1736 by Leonhard Euler, originally entitled 'Solutio problematis ad geometriam situs pertinentis', that appeared in *Commentarii Academiae Scientiarum Imperialis Petropolitanae*

1 In addition to that branch of geometry which is concerned with magnitudes, and which has always received the greatest attention, there is another branch, previously almost unknown, which Leibniz first mentioned, calling it the *geometry of position*. This branch is concerned only with the determination of position and its properties; it does not involve measurements, nor calculations made with them. It has not yet been satisfactorily determined what kind of problems are relevant to this geometry of position, or what methods should be used in solving them. Hence, when a problem was recently mentioned, which seemed geometrical but was so constructed that it did not require the measurement of distances, nor did calculation help at all, I had no doubt that it was concerned with the geometry of position — especially as its solution involved only position, and no calculation was of any use. I have therefore decided to give here the method which I have found for solving this kind of problem, as an example of the geometry of position.

2 The problem, which I am told is widely known, is as follows. In Königsberg in Prussia, there is an island *A*, called the *Kneiphof*, the river which surrounds it is divided into two branches, as can be seen in Figure 1, and these branches are crossed by seven bridges, *a*, *b*, *c*, *d*, *e*, *f* and *g*. Concerning these bridges, it was asked whether anyone could arrange a route in such a way as to cross each bridge once and only once. I was told that some people asserted that this was impossible, while others were in doubt; but nobody would actually assert that it could be done. From this, I have formulated the general problem, whatever be the arrangement and division of the river into branches, and however many bridges there be, can one find out whether or not it is possible to cross each bridge exactly once?

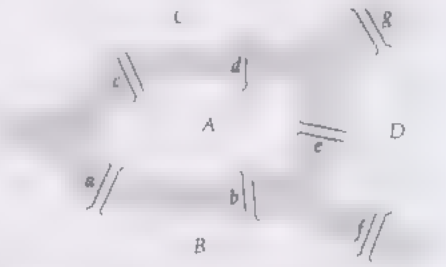


Figure 1

3 As far as the problem of the seven bridges of Konigsberg is concerned, it can be solved by making an exhaustive list of all possible routes, and then finding whether or not any route satisfies the conditions of the problem. Because of the number of possibilities, this method of solution would be too difficult and laborious, and in other problems with more bridges it would be impossible. Moreover, if this method is followed to its conclusion, many irrelevant routes will be found, which is the reason for the difficulty of this method. Hence I rejected it, and looked for another method concerned only with the problem of whether or not the specified route could be found; I considered that such a method would be much simpler.

4 My whole method relies on the particularly convenient way in which the crossing of a bridge can be represented. For this I use the capital letters *A*, *B*, *C*, *D* for each of the land areas separated by the river. If a traveller goes from *A* to *B* over bridge *a* or *b*, I write this as *AB* — where the first letter refers to the area the traveller is leaving, and the second refers to the area arrived at after crossing the bridge. Thus if the traveller leaves *B* and crosses into *D* over bridge *f*, this crossing is represented by *BD*, and the two crossings *AB* and *BD* combined I shall denote by the three letters *ABD*, where the middle letter *B* refers both to the area which is entered in the first crossing and to the one which is left in the second crossing.

5 Similarly, if the traveller goes on from *D* to *C* over the bridge *g*, I shall represent these three successive crossings by the four letters *ABDC*, which should be taken to mean that the traveller, starting

in *A*, crosses to *B*, goes on to *D*, and finally arrives in *C*. Since each land area is separated from every other by a branch of the river, the traveller must have crossed three bridges. Similarly, the successive crossing of four bridges would be represented by five letters, and in general, however many bridges the traveller crosses, the journey is denoted by a number of letters one greater than the number of bridges. Thus the crossing of seven bridges requires eight letters to represent it.

6 In this method of representation, I take no account of the bridges by which the crossing is made, but if the crossing from one area to another can be made by several bridges, then any bridge can be used, so long as the required area is reached. It follows that if a journey across the seven bridges of Figure 1 can be arranged in such a way that each bridge is crossed once, but none twice, then the route can be represented by eight letters which are arranged so that the letters *A* and *B* are next to each other twice, since there are two bridges, *a* and *b*, connecting the areas *A* and *B*; similarly, *A* and *C* must be adjacent twice in the series of eight letters, and the pairs *A* and *D*, *B* and *D*, and *C* and *D* must occur together once each.

7 The problem is therefore reduced to finding a sequence of eight letters, formed from the four letters *A*, *B*, *C* and *D*, in which the various pairs of letters occur the required number of times. Before I turn to the problem of finding such a sequence, it would be useful to find out whether or not it is even possible to arrange the letters in this way, for if it were possible to show that there is no such arrangement, then any work directed towards finding it would be wasted. I have therefore tried to find a rule which will be useful in this case, and in others, for determining whether or not such an arrangement can exist.

8 In order to try to find such a rule, I consider a single area *A*, into which there lead any number of bridges *a*, *b*, *c*, *d*, etc. (Figure 2). Let us take first the single bridge *a* which leads into *A*: a traveller who crosses this bridge must either have been in *A* before crossing, or have come into *A* after crossing, so that in either case the letter *A* will occur once in the representation described above. If three bridges (*a*, *b* and *c*, say) lead to *A*, and if the traveller crosses all three, then in the representation of his journey the letter *A* will occur twice, whether the journey starts from *A* or not. Similarly, if five bridges lead to *A*, the representation of a journey across all of them would have three occurrences of the letter *A*. And in general, if the number of bridges is any odd number, and if it is increased by one, then the number of occurrences of *A* is half of the result.

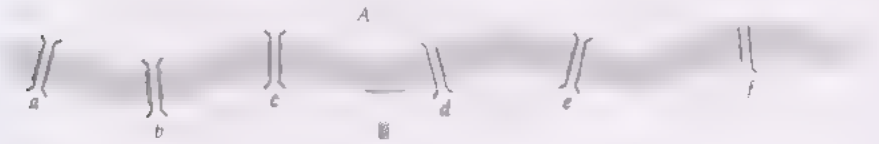


Figure 2

9 In the case of the Königsberg bridges, therefore, there must be three occurrences of the letter *A* in the representation of the route, since five bridges (*a*, *b*, *c*, *d*, *e*) lead to the area *A*. Next, since three bridges lead to *B*, the letter *B* must occur twice; similarly, *D* must occur twice, and *C* also. So in a series of eight letters, representing the crossing of seven bridges, the letter *A* must occur three times, and the letters *B*, *C* and *D* twice each — but this cannot happen in a sequence of eight letters. It follows that such a journey cannot be undertaken across the seven bridges of Königsberg.

Acknowledgements

p.16 title page from *Dissertatio de Arte Combinatoria*, by G. W. Leibniz, courtesy of the British Library;

p.16 title page from *Ars Magna Sciendi sive Combinatoria*, by A. Kircher, courtesy of the British Library;

p.46–47 translation of part of 'Solutio problematis ad geometriam situs pertinentis', by L. Euler, modified from pages 3–5 of *Graph Theory 1736–1936*, by N. L. Biggs, E. K. Lloyd and R. J. Wilson, courtesy of the publishers Oxford University Press, Oxford, UK.

Exercises

Section 1

There are no exercises for Section 1.

Section 2

2.1 Classify each of the following problems as an *existence, construction, enumeration* and/or *optimization* problem.

(a) A manufacturer of tractors can send them to a number of warehouses by means of a number of channels (road, rail, etc.). If there are restrictions on the number that can be sent along each channel each week, what is the maximum number of tractors that can be sent each week?

(b) In how many ways can a floor be tiled using a combination of regular three-sided and four-sided tiles?

(c) How would you design an experiment to compare seven hay-fever drugs?

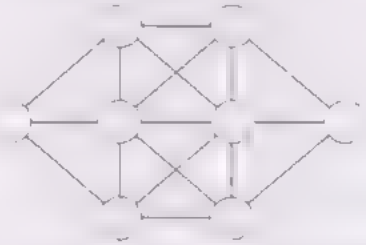
(d) A hiker is planning a trip. There are a number of items the hiker wishes to take along, each of which has a particular value for the trip. But the hiker's knapsack can only accommodate items whose combined weight is less than a certain amount. Which items should be packed so that the total value is a maximum, subject to the weight restriction?

This problem is known as the *knapsack problem*

2.2 Suppose that you want to drive from Land's End to John O'Groats. State the existence, construction, enumeration and optimization problems involved.

Existence/construction problems

2.3 The letters *A, B, C, D, E, F, G, H* are to be put into the circles in the diagram below in such a way that no letter is connected by a line to a letter that is next to it in the alphabet.



(a) Explain why the method of trying all the possibilities (that is, the *exhaustion method*) is not a good approach.

(b) Does a solution exist ? If so, find one; if not, explain why.

Hint Which are the easiest letters to place? Which are the most difficult circles to fill?

2.4 An $n \times n$ **latin square** is a square arrangement of n letters such that each letter appears exactly once in each row and exactly once in each column. For example, a 3×3 latin square is:

$$\begin{array}{ccc} A & B & C \\ C & A & B \\ B & C & A \end{array}$$

(a) Complete the following 4×4 latin square:

$$\begin{array}{cccc} C & - & - & - \\ - & A & - & - \\ - & - & - & B \\ D & - & A & - \end{array}$$

(b) Construct a 5×5 latin square, using the letters A, B, C, D and E.

2.5 An n**-bit Gray code** is a sequence consisting of all the 2^n binary words of length n, arranged in such a way that each word except the first differs in just one bit from the previous word, and the first word differs in just one bit from the last word. For example, a 3-bit Gray code, consisting of eight 3-bit words, is

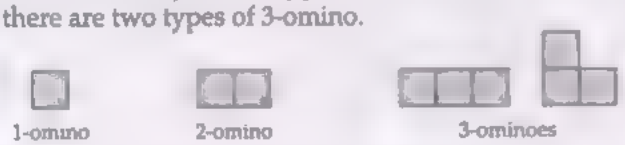
000, 001, 011, 010, 110, 111, 101, 100.

Construct a 4-bit Gray code, consisting of sixteen 4-bit words.

Enumeration problems

2.6 An n**-omino** is an arrangement of n squares of the same size in edge-to-edge contact. There is just one type of 1-omino and just one type of 2-omino, but there are two types of 3-omino.

Two n-ominoes are considered to be the same if one can be placed exactly on top of the other, possibly after turning it over.



How many types of 4-omino are there? Draw them.

2.7

(a) How many ways are there of arranging four books, labelled A, B, C and D, on a shelf?

(b) List all the different arrangements.

2.8 How many ways are there of making up £1 from 10p, 20p and 50p coins? List them.

2.9 In the Morse code, each letter is represented by a sequence of up to four dots and/or dashes, as follows:

A ·–	B –···	C –·–·	D –··	E ·
F ··–·	G ––·	H ····	I ··	J ·–––
K –·–	L ·–··	M ––	N –·	O –––
P ·––·	Q ––·–	R ·–·	S ···	T –
U ··–	V ···–	W ·––	X –··–	Y –·––
Z ––··				

(a) How many other symbols can be represented by up to four dots and/or dashes, by using combinations not used for the letters?

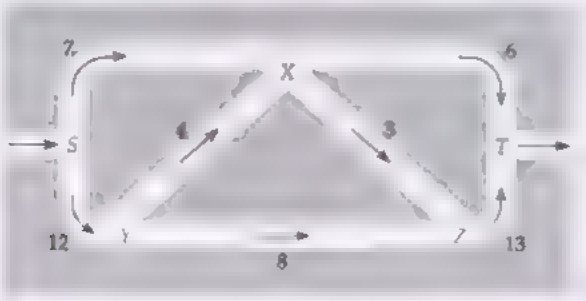
(b) How many symbols can be represented by up to five dots and/or dashes?

2.10 It has been said that a monkey sitting at a keyboard would eventually produce the complete works of Shakespeare. Suppose that a monkey were given a simple keyboard with only 30 keys (26 letters, comma, full stop, space and question mark) and types at random one letter every second. How often, on average, would the monkey type out WILLIAM SHAKESPEARE?

Don't forget the space.

Optimization problem

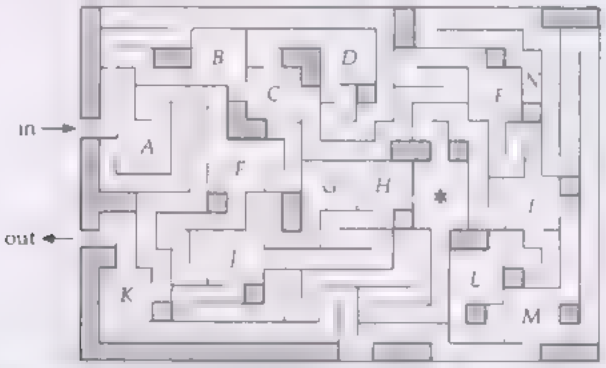
2.11 Use a step-by-step procedure to find the maximum flow of traffic from S to T in the following one-way street system, in which the capacities of the streets are indicated in cars per minute:



Section 3

3.1 Explain how the minimum connector problem in Section 1.9 can be regarded as a problem involving a weighted graph.

3.2 Consider the following maze:

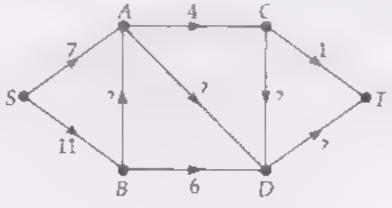


Draw the corresponding graph, and use it to find a way into the centre (*) and a different way out again.

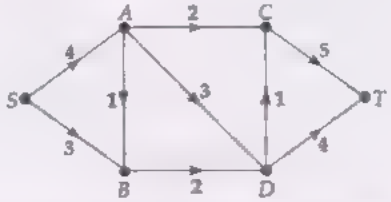
3.3 Draw a digraph to represent the following results of football matches:

Liverpool beat Everton;
Everton beat Arsenal;
Arsenal beat Chelsea;
Everton and Chelsea played each other twice and each won once.

3.4 In the following network, the number next to an arc represents a flow. Use the fact that the total flow into each vertex (other than S and T) must equal the total flow out of it to find the flows in the arcs BA, AD, CD and DT:



3.5 In the following network, the number next to an arc is the capacity of the arc.



An arc that carries a flow equal to its capacity is said to be *saturated*.

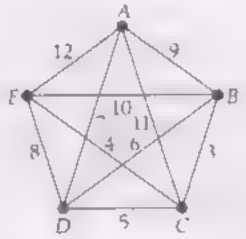
(a) Find a flow from S to T such that the two arcs SA and DT are saturated.

(b) Is there a flow from S to T in which the arc SB is saturated?

Remember, from Section 1.5, that the flow along an arc must not exceed its capacity and that the total flow into a vertex (other than S and T) must equal the total flow out of it

Section 4

4.1 Find a minimum connector for the following graph by using Kruskal's greedy algorithm.



4.2 Find a minimum connector for the graph of Exercise 4.1 by using Prim's greedy algorithm, starting with vertex A.

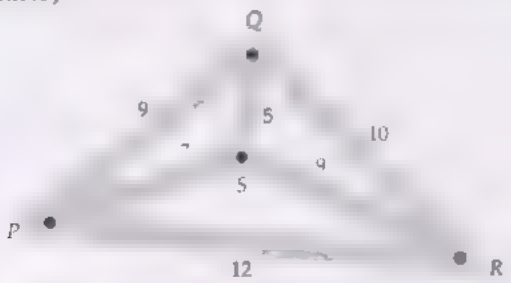
4.3 Use the heuristic algorithm to find an approximate solution to the travelling salesman problem for the graph of Exercise 4.1, starting with the vertex C. Find a better solution by inspection.

4.4 Compare the times taken by an algorithm with time n^{10} and an algorithm with time 10^n, when $n = 5$, 10 and 20, when used with a computer performing a thousand operations per second.

Section 5

5.1 Four housing estates are linked by major roads as follows (distances are marked in miles).

You are advised to watch the television programme associated with Section 5 and to read the corresponding *Television Notes* before attempting this exercise.



(a) Assuming that each housing estate is equally important, what is the best point on the road network to locate a *single* fire station?

(b) How many fire stations are needed so that any housing estate can be reached with 6 minutes by a fire engine travelling at an average speed of 30 miles an hour, and where should they be located?

(c) What are the best locations for:

(1) two fire stations?

(2) three fire stations?

(3) four fire stations?

Solutions to the exercises

2.1

(a) This is an *optimization* problem, since it asks for the maximum number of tractors.

(b) This is an *enumeration* problem.

(c) This is a *construction* problem, since it involves the construction of a suitable experiment. The problem of deciding which is the 'best' hay-fever drug (according to some criterion) may be thought of as an *optimization* problem.

(d) This is an *optimization* problem, since the total value must be a maximum

2.2

Existence	Does there exist a route from Land's End to John O'Groats?
Construction	How do we find such a route?
Enumeration	How many different routes are there, and what are they?
Optimization	Which is the 'best' (shortest, quickest, cheapest, most scenic, ...) route?

2.3

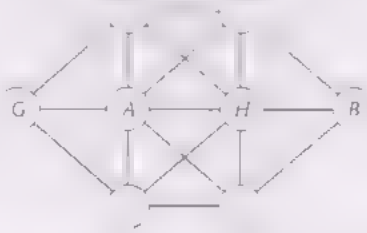
(a) Trying all the possibilities is not a good approach, as the number of different ways of placing 8 letters in 8 locations is 8! = 40 320.

(b) We show that a solution exists by constructing one.

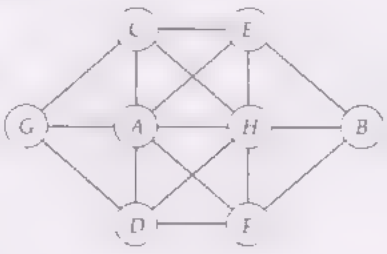
We make the following observations:

- the easiest letters to place are *A* and *H*, because each has only one letter to which it cannot be joined (namely, *B* and *G*, respectively);
- the two circles in the middle are the hardest ones to fill, as each is joined to six others.

This suggests that we place *A* and *H* in the circles in the middle; if we place *A* to the left of *H*, then the only possible places for *B* and *G* are on the far right and the far left, respectively, as follows:



The letter C must now be placed on the left-hand side of the diagram and *F* on the right-hand side, and we can now fill in the remaining letters, as follows:



2.4

(a) There is only one possibility:

C D B A
B A C D
A C D B
D B A C

(b) There are several possibilities — for example:

A B C D E
E A B C D
D E A B C
C D E A B
B C D E A

This example is obtained by moving the letters one place to the right in subsequent rows.

2.5 There are many possibilities — for example:

0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100,
1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000.

2.6 There are five types of 4-omino:

2.7

(a) There are four possibilities for the first book, three possibilities for the second book, two for the third book, and one for the fourth. By the multiplication rule, the total number of different arrangements is $4 \times 3 \times 2 \times 1 = 4! = 24$.

(b) The different arrangements are:

ABCD, ABDC, ACBD, ACDB, ADBC, ADCB, BACD, BADC, BCAD, BCDA, BDAC, BDCA, CABD, CADB, CBAD, CBDA, CDAB, CDBA, DABC, DACB, DBAC, DBCA, DCAB, DCBA.

2.8 The different ways are:

50 + 50,
50 + 20 + 20 + 10,
50 + 20 + 10 + 10 + 10,
50 + 10 + 10 + 10 + 10 + 10,
20 + 20 + 20 + 20 + 20,
20 + 20 + 20 + 20 + 10 + 10,
20 + 20 + 20 + 10 + 10 + 10 + 10,
20 + 20 + 10 + 10 + 10 + 10 + 10 + 10,
20 + 10 + 10 + 10 + 10 + 10 + 10 + 10 + 10,
10 + 10 + 10 + 10 + 10 + 10 + 10 + 10 + 10 + 10.

There are therefore ten ways of making up £1 from 10p, 20p and 50p coins.

2.9

(a) There are:

2 possible letters consisting of one dot or dash;
2^2 possible letters consisting of two dots or dashes;
2^3 possible letters consisting of three dots or dashes;
2^4 possible letters consisting of four dots or dashes.

By the addition rule, the total number of different combinations is

$2 + 2^2 + 2^3 + 2^4 = 30$.

Twenty-six of these are used in the Morse code, and so the remaining four can be used to represent other symbols.

The remaining four are:

(b) The number of different combinations of up to five dots and / or dashes is

$$2 + 2^2 + 2^3 + 2^4 + 2^5 = 62.$$

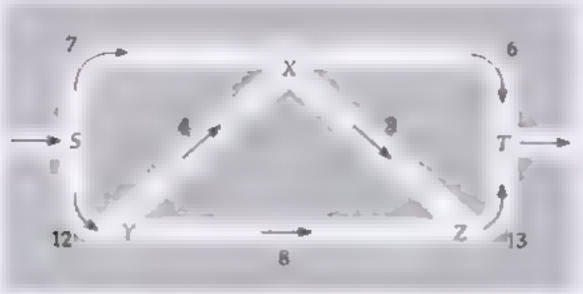
2.10 The number of symbols in WILLIAM SHAKESPEARE is 19 (18 letters and one space) There are 30^{19} ($\approx 1\,16 \times 10^{28}$) possible combinations of 19 letters, and the number of seconds in a year is about 3.15×10^7, so it would take the monkey about

$$(1.16 \times 10^{28} \times 19) / (3.15 \times 10^7) \approx 7 \times 10^{21} \text{ years}$$

to type all possible 19-symbol combinations. Therefore, on average, the monkey would type out WILLIAM SHAKESPEARE once every 7×10^{21} years.

As this number is more than 100 billion times the estimated life of the universe, the monkey would be extremely lucky to produce WILLIAM SHAKESPEARE even once in the universe's estimated lifetime

2.11



First, send 6 cars per minute along the route *SXT*.

Next, send 8 cars per minute along the route *SYZT*.

Next, send 3 cars per minute along the route *SYXZT*.

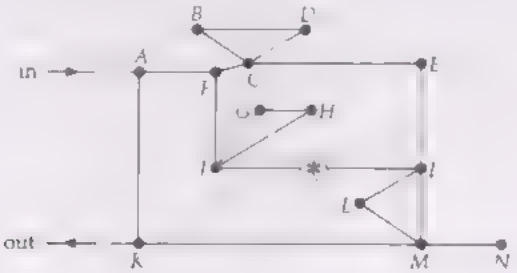
This gives a total flow of traffic from *S* to *T* of 17 cars per minute.

This is clearly the maximum flow from *S* to *T*, since all traffic must pass along one of the roads *XT*, *XZ* or *YZ*, and none of these roads can take any more traffic.

Note that there are other ways of achieving this maximum flow of traffic; in the third step we could have sent 1 car per minute along the route *SXZT* and 2 cars per minute along the route *SYXZT*. for example

3.1 The five vertices correspond to the five towns, and the edges correspond to the possible connections between them. The weight on each edge corresponds to the distance between the corresponding locations. The problem is to find four edges of minumum total weight linking the five vertices together.

3.2 The graph of the maze is

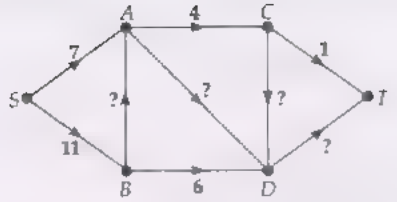


One solution is

$$\text{in} \to A \to F \to J \to * \to I \to M \to K \to \text{out}.$$

3.3 Any of the following digraphs is appropriate:

3.4



The flow along BA is 5 units (consider vertex B: $11 = 6 + ?$).

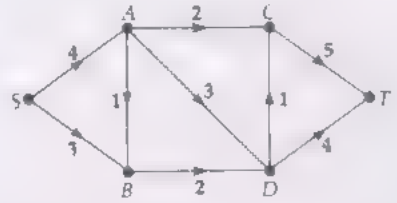
The flow along AD is 8 units (consider vertex A: $7 + 5 = 4 + ?$).

The flow along CD is 3 units (consider vertex C: $4 = 1 + ?$).

The flow along DT is 17 units (consider vertex D: $6 + 8 + 3 = ?$).

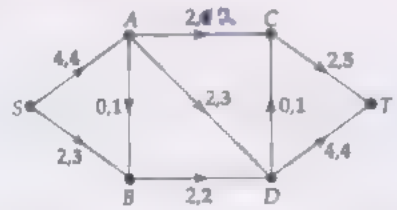
3.5

(a)



If the arcs SA and DT are saturated, then the total flow along each of these arcs must be 4 units. One way of achieving this is to make the flow along each of the arcs AC, AD and BD equal to 2 units and the flow along each of the arcs AB and DC equal to 0. Then the flow along each of the arcs SB and CT must be 2 units. Thus we obtain the following flow from S to T.

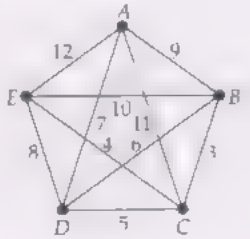
Other solutions are possible



The first number on each arc is the flow, the second (in bold) is the capacity

(b) If the arc SB is saturated, then the total flow into B must be at least 3 units. But there is only one arc directed away from B (namely, BD), and this has a capacity of only 2 units. So the greatest possible flow out of B is 2 units, and so the greatest possible flow into B is also 2 units (or else the flow into B will exceed the flow out of it). Hence, we cannot have a flow from S to T in which the arc SB is saturated.

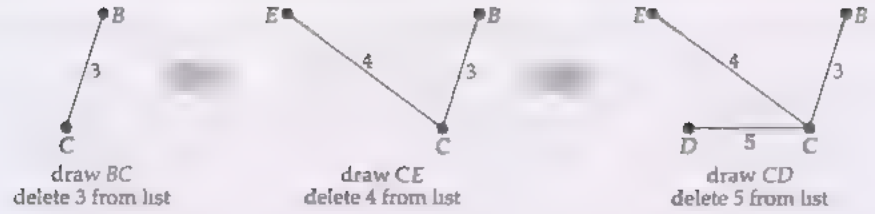
4.1 The given weighted graph is:

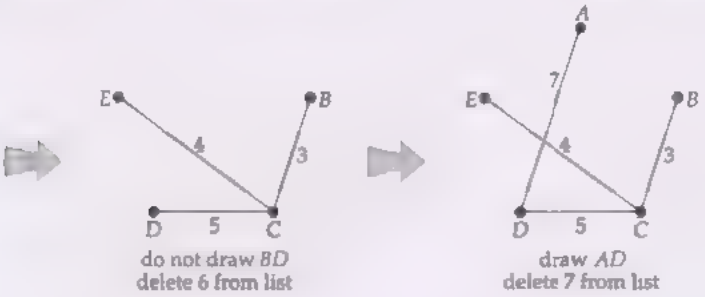


STEP 1 The list of weights, in ascending order, is:

edge	BC	CE	CD	BD	AD	DE	AB	BE	AC	AE
weight	3	4	5	6	7	8	9	10	11	12

STEP 2 The repeated applications of Step 2 may be summarized as follows:



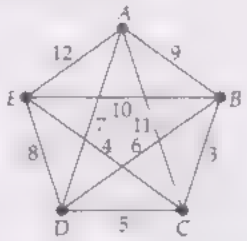


All the vertices are now connected, so we STOP.

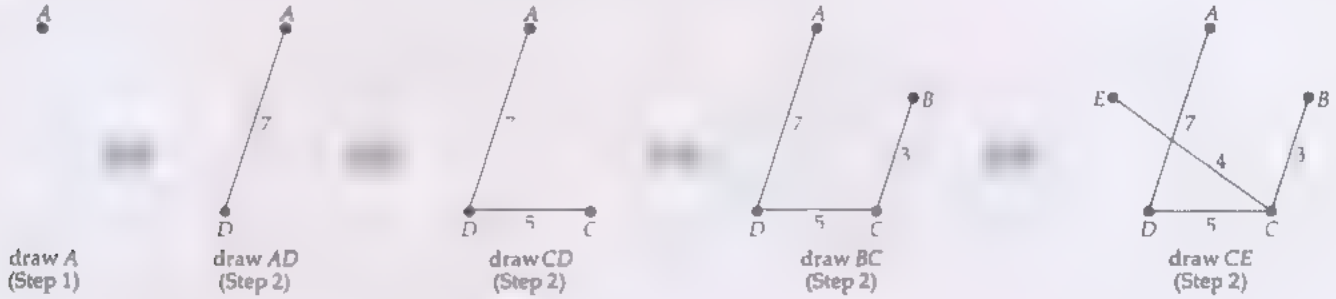
Thus we obtain a minimum connector with total weight

$3 + 4 + 5 + 7 = 19$.

4.2 The given weighted graph is:



The application of Step 1 and the repeated applications of Step 2 may be summarized as follows:

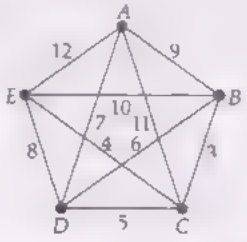


All the vertices are now connected, so we STOP.

Thus we obtain a minimum connector with total weight

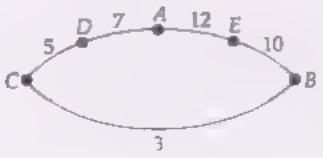
$7 + 5 + 3 + 4 = 19$.

4.3 The given weighted graph is:



The application of Step 1 and the repeated applications of Step 2 may be summarized as follows:

All the vertices now appear in the cycle. We assign the appropriate weight to each edge and STOP.



Thus we obtain the cycle *CDAEBC* with total weight

$5 + 7 + 12 + 10 + 3 = 37$.

A better solution, obtained by inspection, is *CEDABC* with total weight

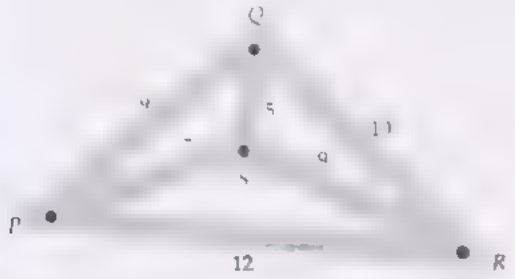
$4 + 8 + 7 + 9 + 3 = 31$.

4.4

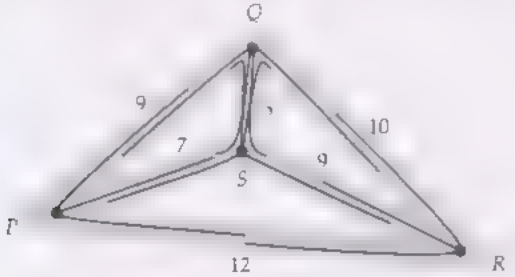
		$n = 5$	$n = 10$	$n = 20$
polynomial time	n^{10}	163 mins	116 days	325 years
exponential time	10^n	1.7 mins	116 days	3.2×10^9 years

Thus, for small values of n, the exponential-time algorithm performs more quickly than the polynomial-time algorithm. It is only for values of n larger than 10 that the polynomial-time algorithm performs more quickly than the exponential-time algorithm.

5.1

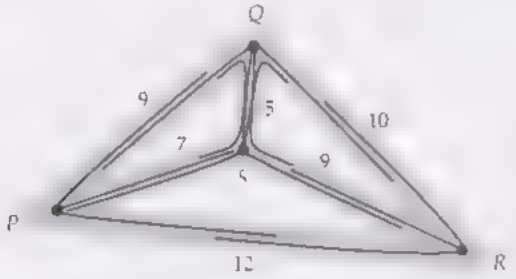


(a) We need to minimize the maximum distance from the fire station to a housing estate. By inspection of the distances on the network, the shortest route from *P* to *R* is along the road *PR*, which has length 12 miles, so that the fire station will have to be at least 6 miles from some of the estates. The points on the road network whose distance is at most 6 miles from a housing estate are shown by the lines penetrating 6 miles from each vertex on the following diagram:



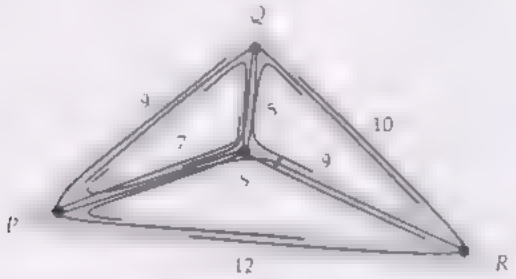
We can see that no point is within 6 miles of all four estates.

Repeating the procedure for 7 miles, we obtain the following diagram:



We can see that no point is within 7 miles of all four estates.

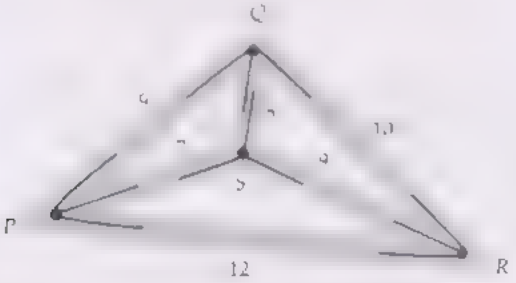
Repeating the procedure for 8 miles, we obtain the following diagram:



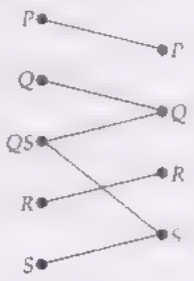
This time the marked point, at one mile along the road SR, is at most 8 miles from each of the four estates, and is the best location for a single fire station.

In the terminology of the television programme, the marked point is the region $PQRS$.

(b) With an average speed of 30 miles an hour, there must be a fire station within 3 miles of each estate if each is to be reached within 6 minutes. The regions P, Q, R and S obtained by penetrating 3 miles from each estate are shown on the following diagram:

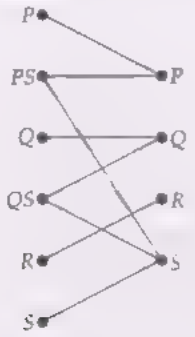


Regions P and R do not overlap with any of the others, but regions Q and S do overlap, along the road QS, creating a new region QS. The corresponding bipartite graph is as follows:



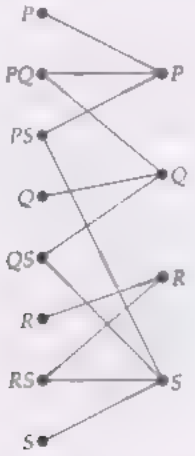
So three fire stations are needed — one in region P (for example, at P itself), one in region R (for example, at R itself) and one in region QS.

(c) (1) For only two fire stations, we need to look for more overlap regions than in part (b). Since the road PS is 7 miles long, a new region PS is created when we penetrate $3\frac{1}{2}$ miles along each road. The corresponding bipartite graph is:



Since no *two* region vertices are together connected to all four estate vertices, in the $3\frac{1}{2}$-mile case we still need three fire stations.

The next new regions PQ and RS were created when we penetrate $4\frac{1}{2}$ miles along each road. The corresponding bipartite graph is:



Since the two region vertices PQ and RS are together connected to all four estate vertices, we can locate our two fire stations in the regions PQ and RS — that is, at the midpoints of the roads PQ and RS.

(2) See part (b).

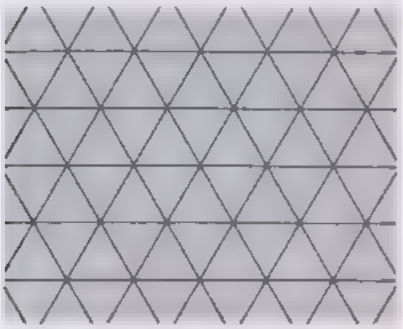
(3) At P, Q, R and S!

Solutions to the problems

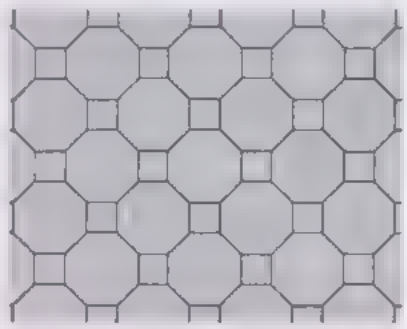
Solution 1.1

No. Three colours are needed for the ring of five states surrounding Nevada, and a different colour must then be used for Nevada. Thus, at least four colours are needed.

Solution 1.2



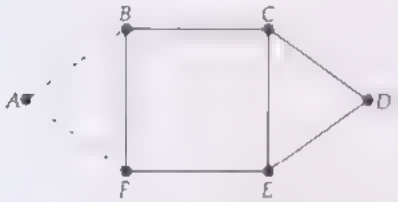
(a)



(b)

Solution 1.3

(a)



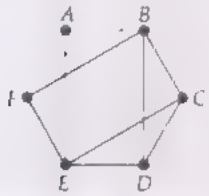
close 2 links (*AB* and *AF*)

close 2 exchanges (*B* and *E*)

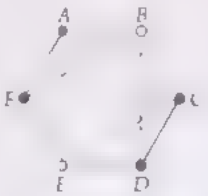
The smallest number of *links* whose closure would separate the network into two parts is 2: *AB* and *AF*, or *BC* and *FE*, or *CD* and *ED*.

The smallest number of *exchanges* whose closure would separate the network into two parts is 2: *B* and *E*, or *B* and *F*, or *C* and *E*, or *C* and *F*.

(b)



close 3 links (*AB*, *AE* and *AF*)



close 2 exchanges (*B* and *E*)

The smallest number of *links* whose closure would separate the network into two parts is 3: *AB*, *AE* and *AF*, or *CB*, *CD* and *CE*, or *DB*, *DC* and *DE*, or *FA*, *FB* and *FE*.

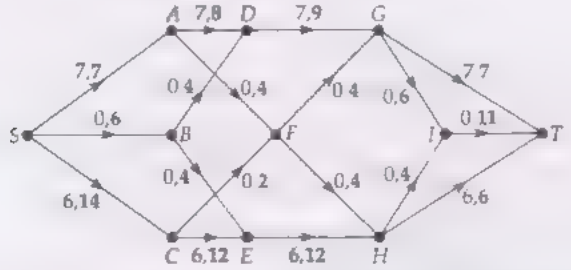
The smallest number of *exchanges* whose closure would separate the network into two parts is 2: *B* and *E*.

Solution 1.4

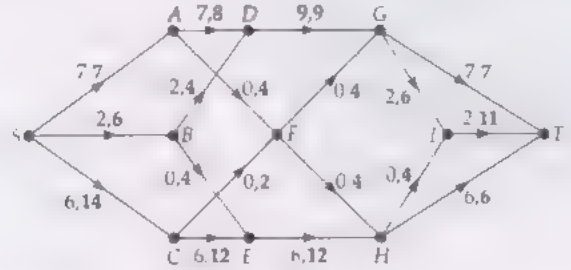
We leave the solution to this problem open for the time being. A solution is given in Section 5, and an explanation of this solution is given in *Graphs 1*.

Solution 1.5

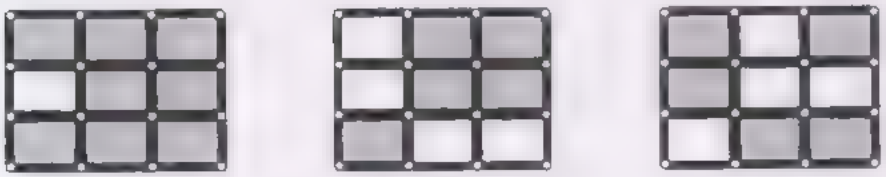
(a) We can, for example, send a flow of 7 units along the route *SADGT* and a flow of 6 units along the route *SCEHT*.



(b) We can, for example, send a further flow of 2 units along the route *SBDGIT*.



Solution 1.6



(a) (b) (c)

(a) One possibility is to start by removing the braces from the top-right and bottom-right rectangles. You can then remove any one other brace, except either of the two in the middle row.

(b) One possibility is to add a brace in the top-left corner.

(c) The framework is rigid, but the removal of any one of the five braces destroys the rigidity.

Solution 1.7

(a) The other solutions are:

1–carpenter,	2–bricklayer,	3–plumber,	4–electrician;
1–carpenter,	2–bricklayer,	3–decorator,	4–electrician;
1–decorator,	2–bricklayer,	3–electrician,	4–carpenter;
1–decorator,	2–bricklayer,	3–plumber,	4–carpenter;
1–decorator,	2–bricklayer,	3–plumber,	4–electrician;
1–decorator,	2–carpenter,	3–plumber,	4–electrician;
1–decorator,	2–plumber,	3–electrician,	4–carpenter.

(b) Yes, it is still possible. All but the last of the above solutions are still valid

Solution 1.8

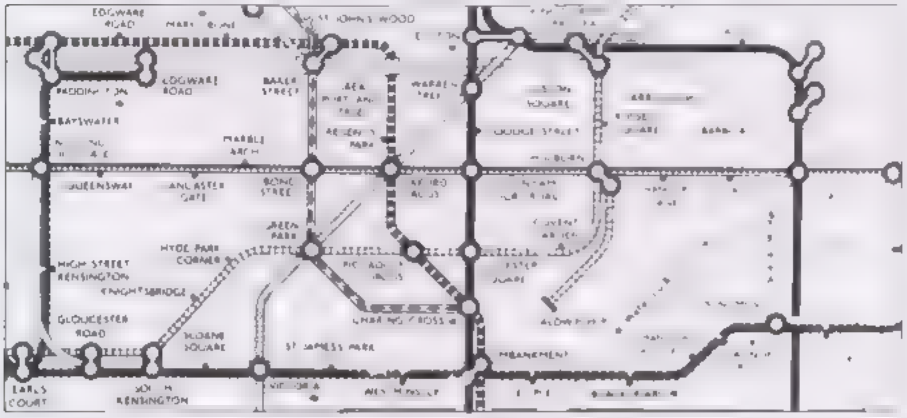
(a) A 'best' route might be one that involves the least number of intermediate stations. Two possible such routes are:

Marble Arch → Bond Street → Green Park →
Charing Cross → Embankment → Westminster;

Marble Arch → Bond Street → Green Park →
Victoria → St James's Park → Westminster.

Alternatively, since changing trains takes time, a 'best' route might be one that involves the fewest number of changes. A route involving only one change is:

Marble Arch → Lancaster Gate → Queensway →
Notting Hill Gate → High Street Kensington →
Gloucester Road → South Kensington → Sloane Square →
Victoria → St James's Park → Westminster.

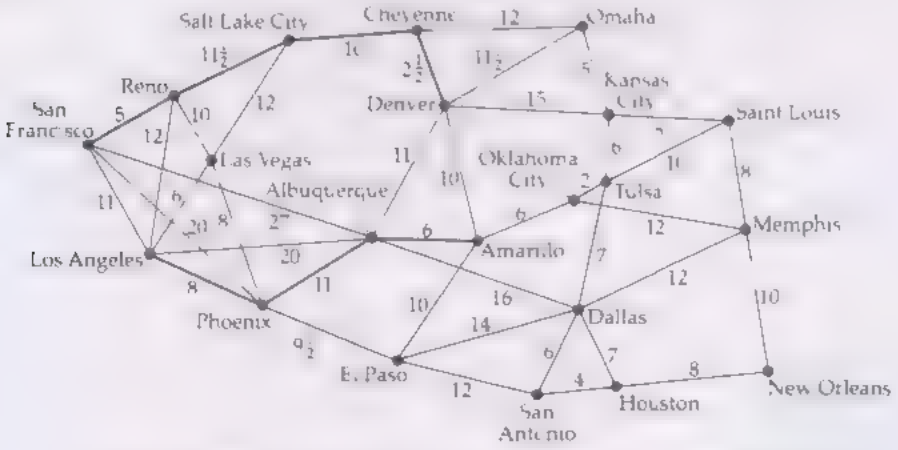


Yet another possibility is to estimate the time taken to travel between adjacent stations and the time to change trains, and then choose the route that involves the least total time. So, for example, if we take the travel time between stations to be 1 minute and the time to change trains to be 3 minutes, then the 'best' route is:

Marble Arch → Bond Street → Oxford Circus → Picadilly Circus →
Charing Cross → Embankment → Westminster.

This route takes 12 minutes (6 stops, 2 changes) compared with 14 minutes (5 stops, 3 changes) for the first two routes above and 13 minutes (10 stops, 1 change) for the third route above.

(b)



The shortest time to travel from Los Angeles to Amarillo is 25 hours, using the route

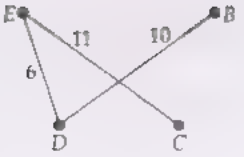
Los Angeles → Phoenix → Albuquerque → Amarillo.

The shortest time to travel from San Francisco to Denver is 29 hours, using the route

San Francisco → Reno → Salt Lake City → Cheyenne → Denver.

A systematic way of solving problems of this type is given in *Networks 2, Optimal paths.*

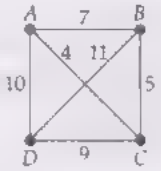
Solution 1.9



(a) A minimum connector that links the towns B, C, D and E comprises the links BD, CE and DE, of total length 27 miles.

(b) A systematic procedure for solving this problem - and indeed all minimum connector problems — is given in Section 4. In that section, you are asked to solve this problem using that procedure.

Solution 1.10

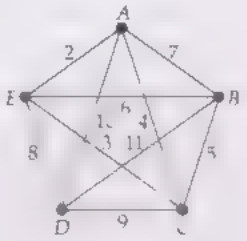


(a) The shortest route is

$$A - C - B - D - A$$

(in either direction) of total length

$$4 + 5 + 11 + 10 = 30 \text{ tens of metres} = 300 \text{ metres.}$$



(b) The shortest route is

$$A - C - B - D - E - A$$

of total length

$$4 + 5 + 11 + 8 + 2 = 30 \text{ tens of metres} = 300 \text{ metres.}$$

Solution 2.1

The answers to this problem are not all clear cut, so do not worry if your answers differ from those given here.

(a) This is an *existence* problem as it stands. Given that such a method exists, actually finding it is a *construction* problem.

(b) This is a *construction* problem.

(c) This is an *optimization* problem, there are many possible routes, and you wish to find the shortest.

(d) This is an *optimization* problem; there are many possible locations, and you wish to find the 'best', according to some criterion.

(e) This is an *enumeration* problem.

(f) This is an *existence* problem as it stands. Given that such a tiling exists, actually finding one is a *construction* problem.

Solution 2.2

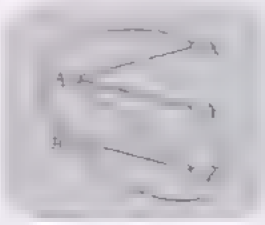
(a) Yes — the route $A \to E \to F \to C \to B$.

(b) No — none of the roads in the one-way system leads into D.

(c) Yes — the route $F \to C \to B \to A \to E$.

Solution 2.3

(a) A suitable repositioning is:

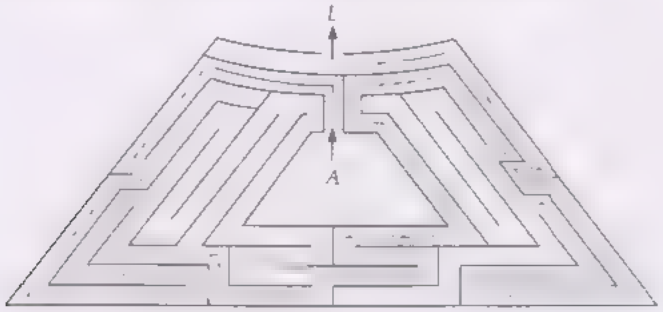


(b) It is not possible to reposition the conducting strips in a way that avoids crossing points.

This is essentially the utilities problem, with gas, water and electricity replaced by X, Y and Z.

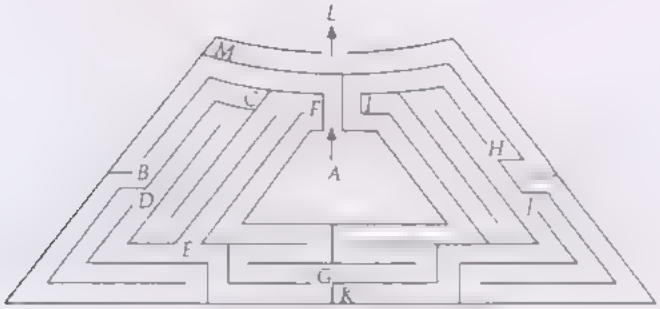
Solution 2.4

A route out of the maze is shown in the following diagram.



Solution 2.5

(a)



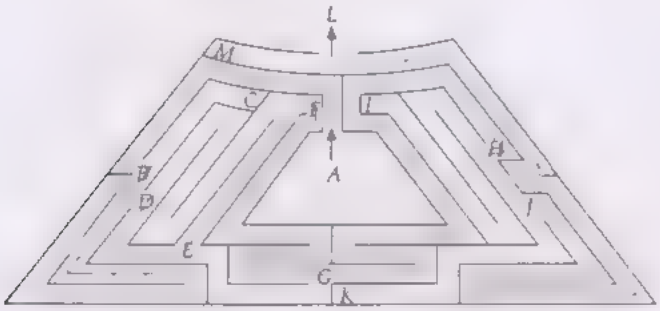
One systematic approach is to record each label as you come to it, to avoid going along any passage more than once unless there is no alternative, and always taking the passage that leads to the next letter in the alphabet if there is more than one choice at any label. This leads to the following record of labels:

$$A \to B \to C \to B \to D \to E \to F \to G \to H \to I \to H \to J \to K \to J \to L.$$

Other systematic approaches are possible

Now, eliminating all the trips to a dead-end and back, we obtain the following route out of the maze:

$$A \to B \to D \to E \to F \to G \to H \to J \to L$$



(b) A possible algorithm, based on the above approach, might be something like the following:

1 Label each junction and each dead-end.

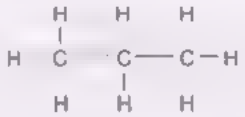
2 At each label, if possible go on to a label you have not been to before.

3 Stop when you reach the exit.

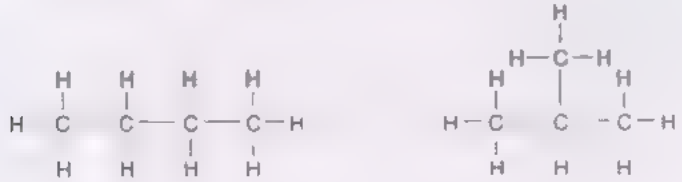
Note that a formal algorithm would have to specify more details than does our informal one here — as you will see in Section 4.

Solution 2.6

For an alkane with formula C_3H_8, the only possible arrangement is as follows:

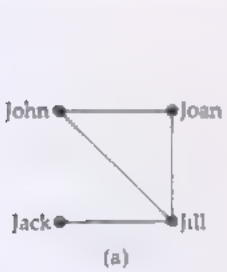


For an alkane with formula C_4H_{10}, the carbon atoms can be arranged in two different ways, as follows:



Solution 3.1

Possible drawings of the graphs of the two situations are:



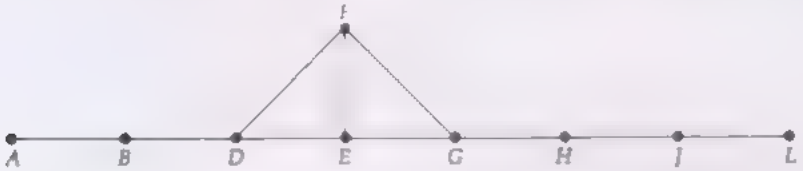
(a)

(b)

Graphs such as graph (b) are called *circulation diagrams* because of their use in analysing the movements of people in large buildings. In particular, they have been used in the design of airports and supermarkets.

Solution 3.2

Because we do not want routes that involve retracing our steps, we can remove the dead-end edges joining B and C, H and I, J and K, and L and M. This gives us the following simplified graph:



Therefore the only four routes out of the maze that do not involve retracing our steps are:

$A \to B \to D \to E \to G \to H \to J \to L$,
$A \to B \to D \to E \to F \to G \to H \to J \to L$,
$A \to B \to D \to F \to G \to H \to J \to L$,
$A \to B \to D \to F \to E \to G \to H \to J \to L$

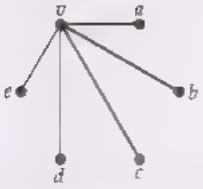
Solution 3.3

We take a vertex corresponding to each person, and join two vertices by a thick edge if the corresponding people know each other and by a thin edge otherwise. We must show that there is always *either* a triangle of thick edges *or* a triangle of thin edges.

Let v be any vertex. Then there must be exactly five edges emerging from v, either thick or thin, and so at least three of these edges must be of the same type.

Let us assume that there are at least three thick edges, as shown in the following diagram:

The case of at least three thin edges is analogous.



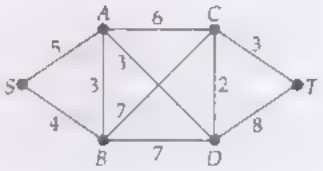
If the people corresponding to vertices a and b know each other, then the edges joining the vertices v, a and b form a triangle of thick lines, as required. Similarly, if the people corresponding to vertices a and c know each other, then the thick edges joining the vertices v, a and c form a triangle, as required, and if the people corresponding to vertices b and c know each other, then the thick lines joining the vertices v, b and c form a triangle.

In the remaining case, a and b do not know each other, a and c do not know each other, and b and c do not know each other, but then the thin lines joining the vertices a, b and c form a triangle, as required.

Solution 3.4

Solution 3.5

The shortest route from A to T is $ADCT$, with length 8; the shortest route from B to T is BCT, with length 10; hence the shortest route from S to T is $SADCT$, with length 13.



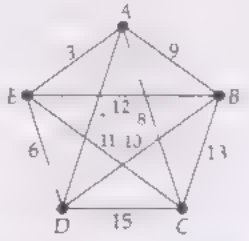
Solution 3.6

(a) Represent the stations by vertices and the railway lines by edges; the weight(s) on an edge might be, for example, the distance between stations, the average time taken to travel between stations, or the number of trains per hour that could run along the corresponding line.

(b) Represent the airports by vertices and the air-routes by edges; the weight(s) on an edge might be, for example, the distance between airports, the average time taken to travel between airports, the number of flights per day possible on that route, the cost of running a flight on that route, the profitability of that route, or the importance of that route to the airline company.

Solution 4.1

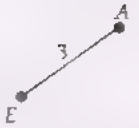
The given weighted graph is:



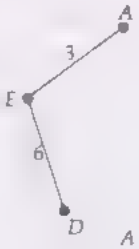
STEP 1 The list of weights, in ascending order, is:

edge	AE	DE	AD	AC	AB	BD	CE	BE	BC	CD
weight	3	6	7	8	9	10	11	12	13	15

STEP 2 The first weight is 3, corresponding to AE. We draw this edge and delete 3 from the list.

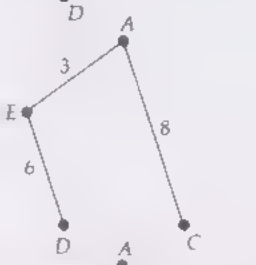


STEP 2 The next weight is 6, corresponding to DE. We draw this edge and delete 6 from the list.



STEP 2 The next weight is 7, corresponding to AD. This would create a cycle, so we just delete 7 from the list.

STEP 2 The next weight is 8, corresponding to AC. We draw this edge and delete 8 from the list.

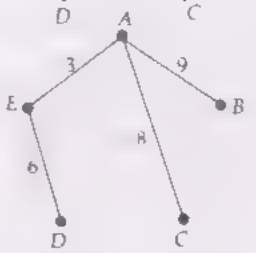


STEP 2 The next weight is 9, corresponding to AB. We draw this edge and delete 9 from the list.

All the vertices are now connected, so we STOP.

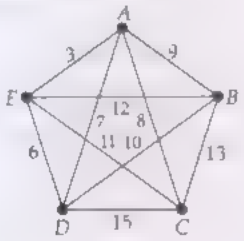
Thus we obtain a minimum connector with total weight

$3 + 6 + 8 + 9 = 26$.

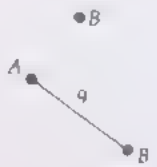


Solution 4.2

The given weighted graph is:

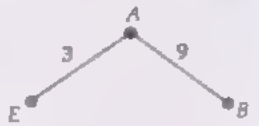


STEP 1 We draw vertex B.

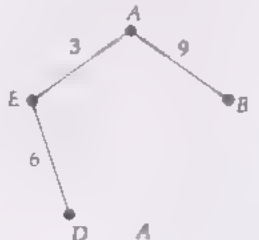


STEP 2 The edge of least weight emerging from B is AB, with weight 9. We draw this edge.

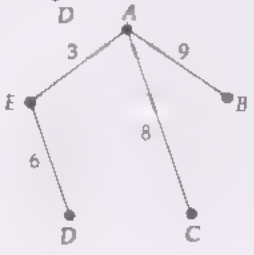
STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is AE, with weight 3. We draw this edge.



STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is DE, with weight 6. We draw this edge.



STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is AC, with weight 8. We draw this edge.



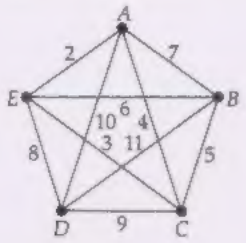
All the vertices are now connected, so we STOP.

Thus we obtain a minimum connector with total weight

$9 + 3 + 6 + 8 = 26$

Solution 4.3

The given weighted graph is:



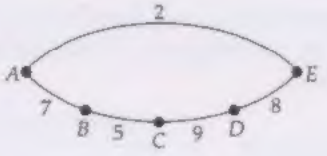
STEP 1 The edge of least weight emerging from A is AE, with weight 2. We draw A and E and join them by two edges, to form a cycle.

STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is CE, with weight 3. We draw C after E in the cycle.

STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is BC, with weight 5. We draw B after C in the cycle.

STEP 2 The edge of least weight joining a drawn vertex to one not currently drawn is DE, with weight 8. We draw D after E in the cycle.

All the vertices now appear in the cycle. We assign the appropriate weight to each edge and STOP.



Thus we obtain the cycle $AEDCBA$ with total weight

$$2+8+9+5+7=31.$$

Solution 4.4

When the algorithm time is n^5, the corresponding time for $n = 10$ is $10^5/1000$ seconds, which is 100 seconds, or 1 minute 40 seconds.

When the algorithm time is 5^n, the corresponding time for $n = 10$ is $5^{10}/1000$ seconds, which is approximately 9766 seconds, or just over 2.7 hours.

Solution 5.1

We wish to solve the following problem:

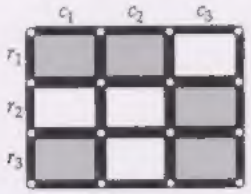
is the given braced rectangular framework rigid?

We identify the following important features:

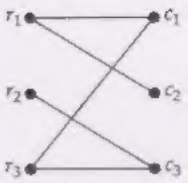
- the framework consists of rows and columns of rectangles;
- some of the rectangles are braced.

We introduce some notation for labelling the rows (r_1, r_2, r_3, ...) and columns (c_1, c_2, c_3, ...), so that we can specify which rectangles are braced. By introducing this notation, we can represent a braced rectangular framework by a graph, in which the vertices represent the rows and columns and the edges represent the braces.

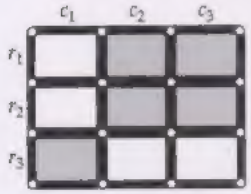
Next, we investigate some examples:



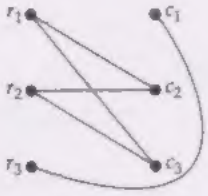
rigid bracing



graph in one piece



non-rigid bracing



graph in two pieces

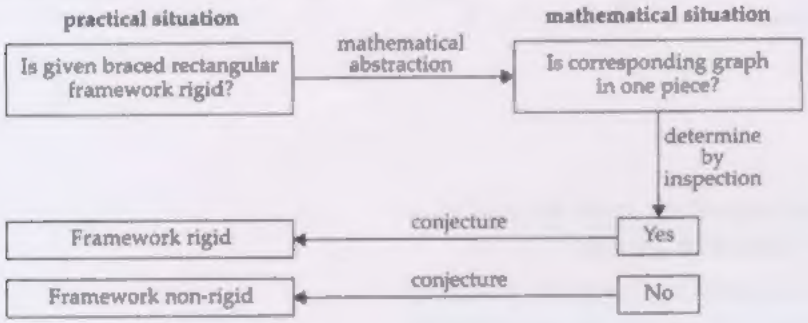
As a result of investigating various examples, we arrive at the following conjectures:

- if a braced rectangular framework is rigid, then the graph is in one piece;
- if the graph of a framework is in one piece, then the framework is rigid;
- if a braced rectangular framework is not rigid, then the graph comprises two or more distinct pieces;
- if the graph of a framework comprises two or more distinct pieces, then the framework is not rigid.

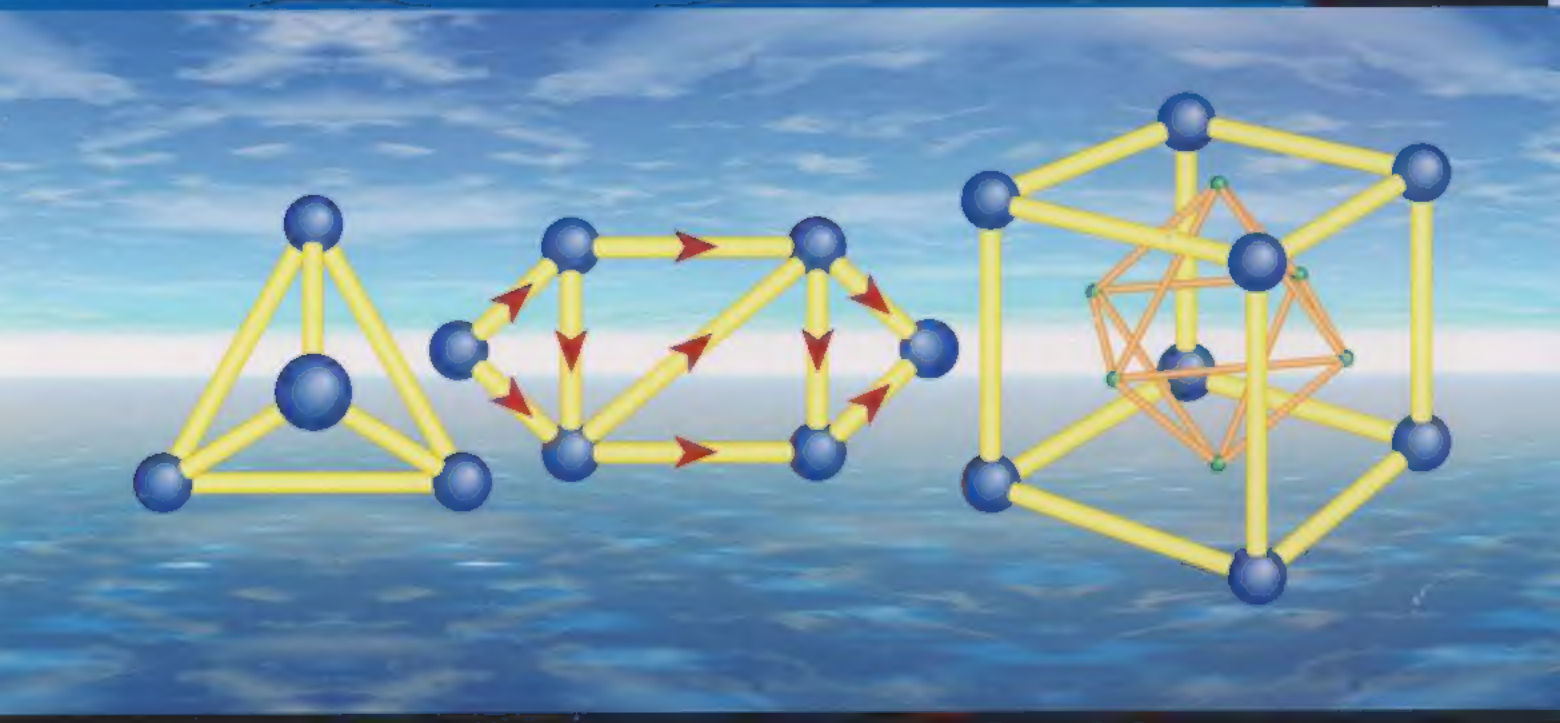
If you did not arrive at such conjectures as a result of your computer work on braced rectangular frameworks, you may like to try the related activities again.

At this stage in the course, we are unable to *prove* these conjectures, so the mathematical modelling process is incomplete. Nonetheless, we can summarize the above stages as follows:

The conjectures will be proved in *Design 2, Kinematic design*.



Index



MT365 Graphs, Networtks and Design

MT365 Introduction
ISBN 0 7492 2224 7